Clustering Techniques in Speaker Recognition

THESIS

Douglas Neale Prescott
Flight Lieutenant, Royal Australian Air Force

AFIT/GE/ENG/94M-05.

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

Best
Available
Copy

AFIT/GE/ENG/94M-05.

Clustering Techniques in Speaker Recognition

THESIS

Douglas Neale Prescott
Flight Lieutenant, Royal Australian Air Force

AFIT/GE/ENG/94M-05.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE
March 1994

3. REPORT TYPE AND DATES COVERED
Master's Thesis

4. TITLE AND SUBTITLE
Clustering Techniques for Speaker Recognition

5. FUNDING NUMBERS

6. AUTHOR(S)
D. Neale Prescott

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GE/ENG/94M-05

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Lt Col Rodney Winter
NSA/R221
Fort Meade, MD 20755-6000

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Distribution Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This thesis presents a comparison based on identification rate, of three clustering techniques applied to cepstral features for speaker identification. LBG vector quantization as developed by Linde, Buzo and Gray; is used to provide benchmark performance for comparison with Fuzzy clustering (based on the un-supervised fuzzy partition-optimal number of classes, UFP-ONC algorithm by Gath and Geva) and an Artificial Neural Network, the Multilayer Perceptron.
Cepstral features from the TIMIT, King and AFIT93 corpus speaker databases are used to produce speaker-identification classifiers using each of the clustering algorithms. The experiment reported evaluates the speaker identification performance using the 20-dimensional cepstral features which were extracted directly from the databases. The speaker databases were taken from different recording environments, TIMIT is studio quality, AFIT93 was recorded in an office environment and King is recorded telephone conversations. The performance provides an indication of merit for the clustering techniques for the range of typical recording environments. This thesis demonstrates the application of fuzzy clustering for speaker identification. It is shown that the UFP-ONC algorithm can achieve identification rates equal to the LBG vector quantization system. LBG vector quantization provides the best overall performance of all three clustering techniques.

14. SUBJECT TERMS
Pattern Recognition, Recognition, Identification, Biometry

15. NUMBER OF PAGES
89

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

AFIT/GE/ENG/94M-05.

Clustering Techniques in Speaker Recognition

THESIS

Presented to the Faculty of the Graduate School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Electrical Engineering

Douglas Neale Prescott, B.Eng
Flight Lieutenant, Royal Australian Air Force

March, 1994

Approved for public release; distribution unlimited

Acknowledgements

There are many people to thank for their kindness, support and knowledge, without which my work would have floundered.

To my wife, Margaret, thank you for your love, trust and courage while I raced off like *Don Quixote* to a windmill half way around the world. To our families for their support, and knowing to telephone in the darkest hours of this endeavour. Your encouragement and caring means a lot to both of us.

To my thesis committee, Doctor Steven Rogers, Captain Dennis Ruck and Doctor Mark Oxley thank you for your enthusiasm, patience and tutoring.

To the *Pattern Wreckers* Curtis Martin, Kim McCrae, Bob MacDonald, John Keller, Martin Chin and John Colombi whose friendship and teamwork made this great experience.

Douglas Neale Prescott

Table of Contents

List of Figures

List of Tables

AFIT/GE/ENG/94M-05.

Abstract

This thesis presents a comparison based on identification rate, of three clustering techniques applied to cepstral features for speaker identification. LBG vector quantization as developed by Linde, Buzo and Gray; is used to provide benchmark performance for comparison with Fuzzy clustering (based on the un-supervised fuzzy partition-optimal number of classes, UFP-ONC algorithm by Gath and Geva) and an Artificial Neural Network. the Multilayer Perceptron.

Cepstral features from the TIMIT, King and AFIT93 corpus speaker databases are used to produce speaker-identification classifiers using each of the clustering algorithms. The experiment reported evaluates the speaker identification performance using the 20-dimensional cepstral features which were extracted directly from the databases. The speaker databases were taken from different recording environments, TIMIT is studio quality, AFIT93 was recorded in an office environment and King is recorded telephone conversations. The performance provides an indication of merit for the clustering techniques for the range of typical recording environments. This thesis demonstrates the application of fuzzy clustering for speaker identification. It is shown that the UFP-ONC algorithm can achieve identification rates equal to the LBG vector quantization system. LBG vector quantization provides the best overall performance of all three clustering techniques.

Clustering Techniques in Speaker Recognition

I. Introduction

1.1 Background

Ensuring that only the right people have access to buildings, financial records and computer systems requires an effective identification system. Traditional systems using locks, combinations, passwords and identification cards can be compromised by copies, decoding or theft of the access device.

Automatic identification of people based on physical features of speech patterns, fingerprints, faces, blood vessels patterns in the retina irises and DNA are all being researched to provide better security for both the individual and organizations.

Automatic speaker recognition determines the identity of a person from a known population of speakers [38] . In this work Cepstral [37] features provide the patterns which are matched to speaker dependent templates.

Potential applications of speaker identification/verification systems include

- Building access systems,
- Secure access to computer records,
- Covert detection of criminal activity on telephone systems,
- Verification of instructions over communications systems, for both military and commercial applications.
- Automatic message routing

1.2 Problem

This work examines the use of clustering techniques using cepstral coefficients for speaker identification. A comparison between vector quantization, fuzzy clustering and an artificial neural network (ANN) is made.

The speech data utilized are a combination of phonetically balanced and text-independent sentences of various lengths. Three databases provides different background environments from pristine [TIMIT], computer room [AFIT corpus 93] [11] and telephone [King]. The AFIT corpus was initiated in 1992 by Colombi [11] for speaker identification, AFIT93 corpus was recorded by the author during this thesis research.

1.3 Scope

This research will implement the Unsupervised Fuzzy Partition-Optimal Number of Classes, clustering algorithm [19], and evaluate its performance for speaker identification. The UFP-ONC will be compared against vector quantization (VQ)[23] and the Multilayer Perceptron using three speech databases. The three databases are the King [25], TIMIT [36], and an AFIT corpus database collected during this research.

1.4 Assumptions

It is assumed that cepstral processing of speech provides the necessary features to uniquely define a speaker. Throughout this work the speech recordings are assumed to be of finite length, in English, and contain only one speaker. Any noise inherent in the databases will be considered typical and no noise reduction techniques will be applied.

1.5 Thesis Organization

Chapter II is a review of relevant literature relating to Fuzzy clustering, vector quantization, neural networks and speech processing. Chapter III presents the methodology used and the experiments conducted. Chapter IV presents the results for each of the three techniques, vector quantization, Fuzzy k-Means and Artificial Neural Networks applied to speech for speaker identification.

II. Literature Review

2.1 Introduction

Speech processing brings together the different fields of signal processing, physics, pattern recognition, linguistics, physiology, psychology, computer science and information theory [40, 42].

In general, speaker identification methods attempt to isolate acoustic features which are dependent on the individuals vocal tract. Formant frequencies [22], formant ratios [39] and pitch contours [4] are examples of features which have been used in speaker identification. Automatic classification of speakers, requires partitioning the feature space in a way that will separate each of the individuals.

This chapter presents a review of speaker identification, cepstral features, vector quantization, fuzzy clustering and artificial neural network literature. For comprehensive reviews on each of these fields refer to references [6, 8, 11, 20, 23, 26, 31, 35, 38]

2.2 Speaker Identification

The task of speaker identification (SID) is to find the absolute identity of a person based only on their speech. The speech features from the subject are compared to each of the people in a known population. O'Shaughnessy [38] provides a review of speaker identification and verification. Speaker Verification is considered the easier of the two problems [38, 40]. The subject makes a claim that he is speaker X, and the verification system makes a binary decision, yes or no , by comparing the subject's speech features with the templates of speaker X. Speaker identification requires comparison with all known speakers and selects the best match. O'Shaughnessy discusses one of the inherent problems in speaker identification, the larger the population of known speakers the more difficult the process becomes. Assuming equal a priori of speakers, the probability P_x that a subject is known is $P_x = 1/N$, where N is the number of speakers in the database. Jayant [35] discusses the computational cost of speaker identification, and notes that a simple system must perform N decisions to decide that the subject is a known speaker. More efficient search algorithms can reduce this by grouping speakers into categories such as male/female,

thus narrowing the search area. In "Statistical Techniques for Talker Identification" [9] Bricker et al. provide insight into the difficulties presented by large databases of speakers. When the speaker database is relatively small, perhaps less than 50, a codebook for each speaker can be kept. However, when the number of speakers in the database becomes large there is immense overhead in storage and search times. To overcome this the authors demonstrate the use of a *nearest neighbor* classifier.

Applications of SID are numerous, such as controlling access to buildings, controlling access to privileged information on computer files or by telephone. Several authors have presented papers on the applications for speaker identification, for a recent cross-section refer to [16, 35, 38, 45, 49, 50] .

2.3 Cepstral Features

The *cepstrum* (or power cepstrum) provides a method of separating the vocal tract spectrum from the spectrum of the vocal tract excitation. Using the cepstrum of speech we attempt to characterize an individual by their long term vocal tract spectrum. The definition of cepstrum is shown below. The Fourier Transform of a time function is denoted by

$$Y(\omega) = \mathcal{F}\{y(t)\}$$

The power spectrum is

$$Y_p(\omega) = |Y(\omega)|^2$$

The cepstrum being

$$C(\tau) \equiv \mathcal{F}\{\log Y_p(\omega)\} \tag{1}$$

The power spectrum is symmetric and real; accordingly we can expand the cepstrum as a Fourier series

$$\log Y_p(\omega) = \sum_{n=-\infty}^{\infty} c_n e^{-jn\omega} \tag{2}$$

where $c_n = c_{-n}$ are real and referred to as cepstral coefficients.

Consider a speech signal, $y(t)$, with the power spectrum, $Y_p(\omega)$. Assuming the vocal tract to be a linear system, we can separate the excitation spectrum $X(\omega)$, and the vocal tract spectrum $H(\omega)$ as shown below. The assumption of stationarity holds for short periods, typically 40 milliseconds [42].

Figure 1 shows each of these steps applied to a segment of the vowel IY, as shown in the upper left corner. The signal is 256 samples, sampled at 8000 samples per second. The magnitude of the Fourier transform is displayed in the upper right. Note the large amplitude components below 200Hz, the glottal pitch and first formant are very distinct approximately 85Hz and 125Hz respectively. The higher frequency formants can be seen in the interval between 2000Hz and 4000Hz. It is difficult to separate the excitation, glottal pulse frequency, from the vocal tract spectrum using the Fourier transform alone. The lower right of Figure 1 shows the logarithm of the Fourier transform, or power spectrum. More detail of the spectrum is now evident, although the glottal frequency still dominates the lower frequencies. The lower left plot displays the cepstrum of the speech, the glottal pitch is now clearly evident at approximately 11.5 (quefrency). The vocal tract information is located in the interval between zero and five. In a discussion of feature extraction for speech recognition (where identity is a secondary concern) Rabiner and Juang discuss *liftering*, or normalizing the cepstrum to remove these low cepstral coefficients which are, "due to variations in transmission, speaker characteristics, vocal effects . . . ", [42], this is the information speaker-identification systems seek to exploit.

Consider the power spectrum signal

$$Y_p(\omega) = X(\omega)H(\omega)$$

Where $H(\omega)$ is the vocal tract transfer function and $X(\omega)$ is the excitation function, taking the logarithm

$$\log[Y_p(\omega)] = \log[X(\omega)] + \log[H(\omega)]$$

followed by the Fourier Transform

$$\mathcal{F}\{\log[Y_p(\omega)]\} = \mathcal{F}\{\log[X(\omega)]\} + \mathcal{F}\{\log[H(\omega)]\} \qquad (3)$$

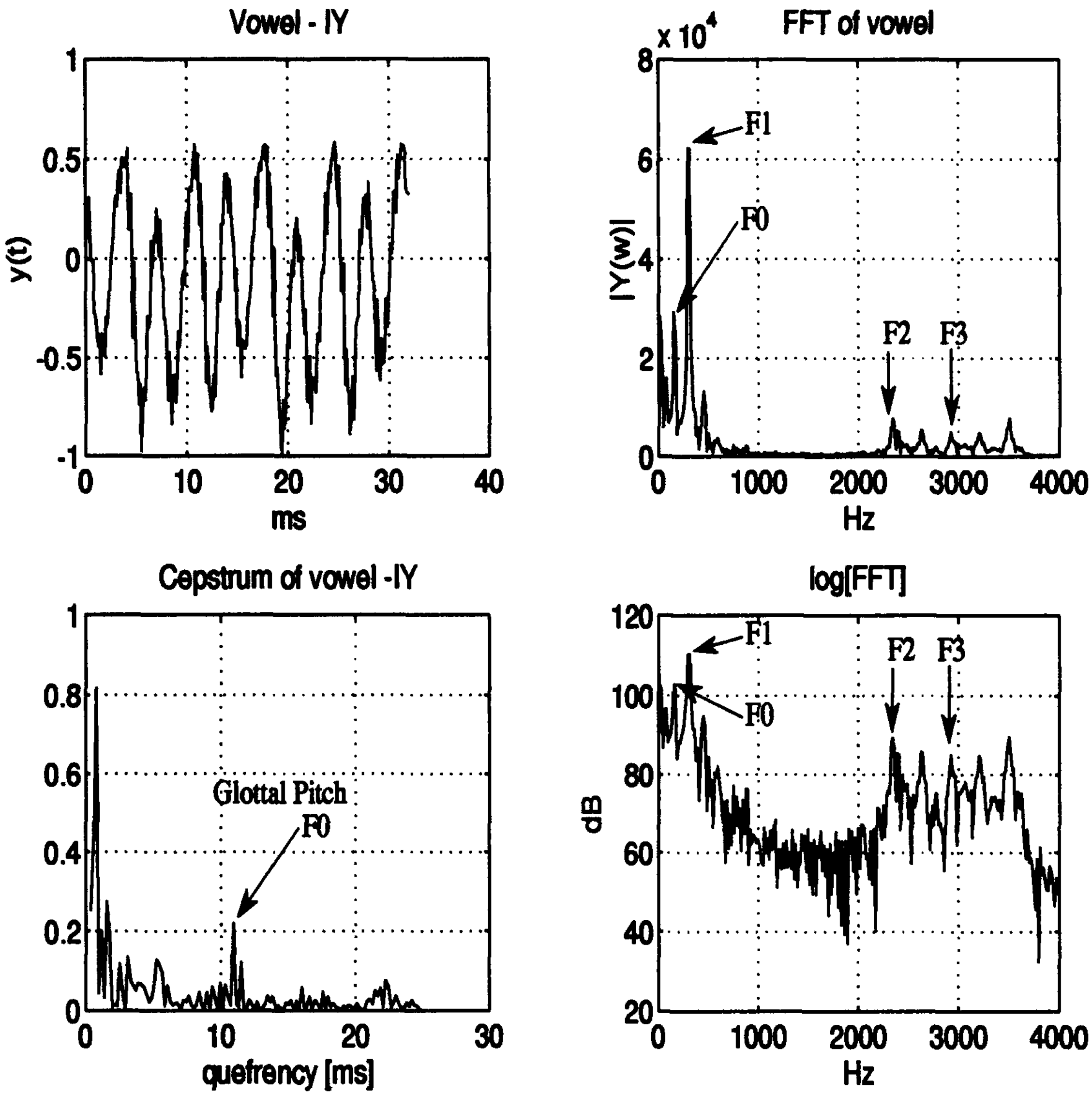


Figure 1. Vowel signal, spectrum and cepstrum

we obtain the cepstrum, which is comprised of two components, the excitation and the vocal tract transfer function. As shown in Figure 1, the cepstrum is very effective at separating the pitch from the vocal tract spectra when using voiced speech. Estimation of the vocal tract transfer function is more difficult during unvoiced speech. Unvoiced excitation is then modelled as a pseudo random process. Parsons [40:pp.120] discusses some models of noise processes associated with different acoustic phonetics.

Cepstral coefficients can be obtained from both the Fourier transform (FFT) and Linear Predictive coding (LPC). Furui compared both features for speaker identification [18] and concluded both achieved the same performance, although LPC-cepstrum is much faster to compute.

As the cepstral coefficients used in this thesis are derived from the LPC-cepstrum, a brief explanation of Linear predictive coding, the use of an all-pole model and the LPC-cepstrum follows. It is generally accepted that an all-pole model of the vocal tract is used in linear predictive coding [40], because of its simplicity, and the ability to model zeroes introduced by the nasal tract. Parsons [40] discusses the conditions for approximating a zero by a number of poles. Increasing the order of the all-pole model provides an effective and tractable model of the human vocal tract. The derivation of the LPC-cepstrum is based on this all-pole model. The aim of linear prediction is to estimate the output of a linear time-invariant system based on past values of input and output.

$$\hat{y}[n] = \underbrace{\sum_{j=0}^{q} b[j]\, x[n-j]}_{\text{current and previous inputs}} - \underbrace{\sum_{i=1}^{p} a[i]\, y[n-i]}_{\text{previous outputs}} \qquad (4)$$

Where $\hat{y}[n]$ is the predicted output, $a[i]$ and $b[j]$ are the predictor coefficients. When y converges we have

$$\sum_{i=0}^{p} a[i]\, y[n-i] = \sum_{j=0}^{q} b[j]\, x[n-j]$$

Taking the z-transform

$$Y(z) \sum_{i=0}^{p} a[i]\, z^{-i} = X(z) \sum_{j=0}^{q} b[j]\, z^{-j}$$

Taking the ratio of outpu to input

$$\frac{Y(z)}{X(z)} = \frac{\sum_{j=0}^{q} b[j]\, z^{-j}}{\sum_{i=0}^{p} a[i]\, z^{-i}}$$

This is the vocal tract spectrum, $H(z)$

$$H(z) = \frac{\sum_{j=0}^{q} b[j]\, z^{-j}}{\sum_{i=0}^{p} a[i]\, z^{-i}}$$

By using an all pole model the numerator reduces to a gain term

$$H(z) = \frac{\sigma}{\sum_{i=0}^{p} a[i]\, z^{-i}}$$

The output is now a function of previous outputs only, and setting $a[0] = 1$

$$H(z) = \frac{\sigma}{1 - \sum_{i=1}^{p} a[i]\, z^{-i}} \tag{5}$$

Where we redefine $a[i]$ to be $-a[i]$ for convenience. This is usually represented as

$$H(z) = \frac{\sigma}{A(z)}$$

Where $A(z) \equiv \sum_{i=0}^{p} a[i]\, z^{-i}$ To compute the cepstral coefficients from the LPC model the logarithm of the transfer function is taken

$$\log[\sigma/A(z)] = \log\sigma + \sum_{n=1}^{\infty} c_n z^{-n}$$

differentiating with respect to z^{-1}, and making a Taylor series expansion

$$c_n = -a_n - \frac{1}{n}\sum_{k=1}^{n-1} k c_k a_{n-k} \quad \text{for } n > 0 \tag{6}$$

where $a_0 = 1$, and $a_k = 0$ for $k > p$.

The log power spectrum becomes

$$\log\left[\sigma^2/|A(e^{j\omega})|^2\right] = \sum_{n=-\infty}^{\infty} c_n e^{-jn\omega} \tag{7}$$

where , $c_n = c_{-n}$, are the LPC-cepstral coefficients.

The cepstral features of a vowel IY are shown in Figure 1, the arrows indicate the glottal pitch and the first three formants. Formants are the resonant frequencies of the vocal tract due to the glottal pulses [40]. The cepstral features can be used to track the pitch and to characterize the vocal tract transfer function of a speaker [38, 40].

Figure 2 shows the cepstral features of two speakers from King . The figure shows the feature vectors of voiced speech during a 60 second conversation. While the text of the speech is not identical, there are still visible differences between the speakers. The purpose of applying clustering algorithms to these features is to characterize each speaker by the uniqueness of the patterns evident in Figure 2

2.4 Vector Quantization

Vector quantization is procedure for representing a signal by a number of symbols or codewords. Vector quantization is widely used in communication systems, where an analog input is transmitted as a sequence of binary codes. Reconstruction of the signal is achieved using the codebook in reverse, mapping the codewords into an analog signal. The objective of vector quantization is to determine the optimal codebook that ensures reconstruction with minimal distortion. Gersho and Cuperman [20] discuss the applications of VQ to speech transmission including different codebook design methods. There are a number of ways to design VQ codebooks, in this research the technique developed by Linde, Buzo and Gray [30] is used. For an excellent review on vector quantization and codebook design

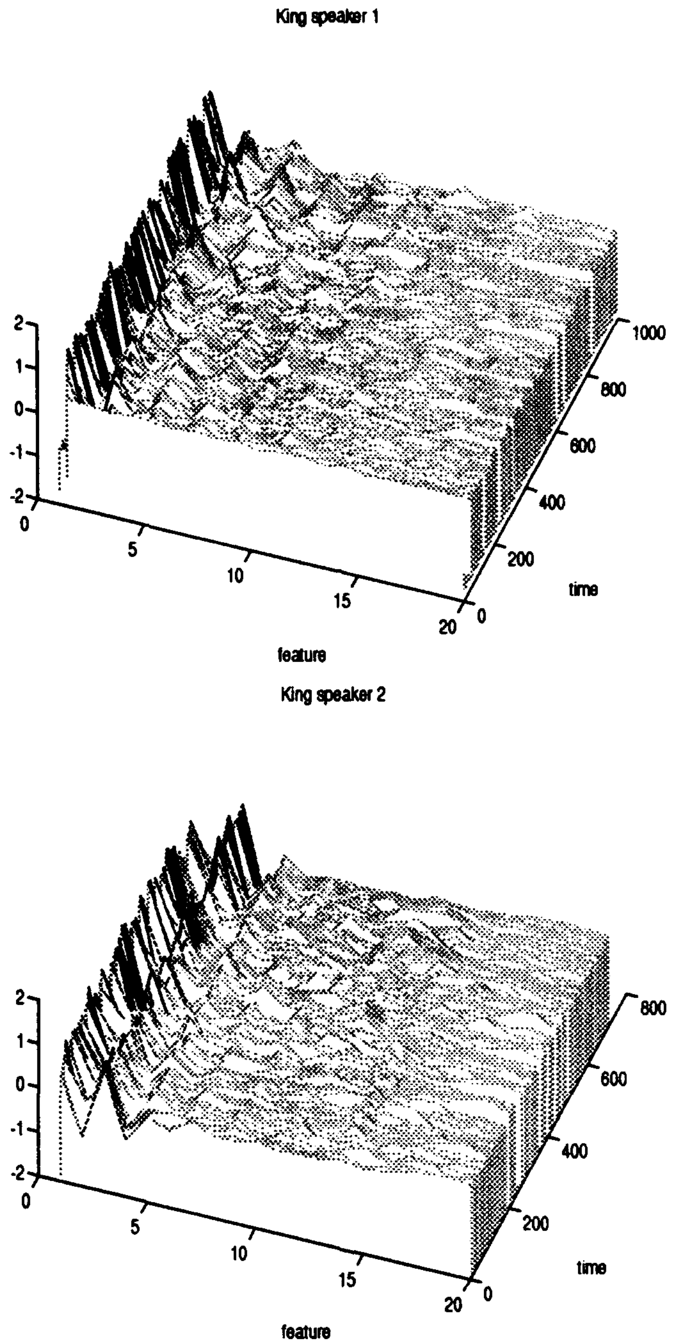


Figure 2. Cepstral feature vectors from King speakers One and Two. (60 seconds of conversation from Session one)

the reader should refer to the 1984 paper by Gray [23]. The Linde, Buzo and Gray (LBG) technique forms a codebook by progressive splitting of codewords. The LBG algorithm is implemented as follows :

1. Find the mean of the data set, use this as the initial k-dimensional codeword, y_0,
2. Double the size of the codebook by splitting each existing codeword
   - $y_n^+ = y_n(1+\epsilon)$,
   - $y_n^- = y_n(1-\epsilon)$,
   - where, n, ranges from 1 to the current size of the codebook,
   - $(0.01 \leq \epsilon \leq 0.05)$ is a scalar multiplier.
3. Iteratively recalculate the centroids, using k-means, to obtain the optimal centroid locations.
4. Repeat steps two and three until the codebook is the desired size.

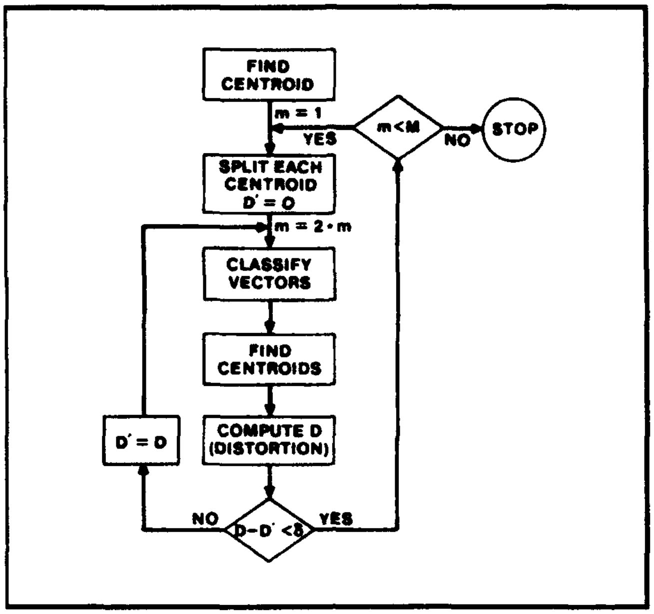


Figure 3. Vector Quantization codebook design algorithm [42]

A flow chart of the algorithm is shown in Figure 3. The **Classify Vectors** is the nearest-neighbor algorithm [17] (k-nn) and the **Find Centroids** is the k-means algorithm [47].

A simple example of codebook splitting using a 2-dimensional data set is illustrated in Figure 4. Figure 4(a) shows an initial codeword, the mean. The codebook is split two, the LBG algorithm computed and the two new codewords are shown in 3(b). A final splitting produces four codewords which represent the four classes. In order to terminate with these four codewords the user must know that only four centers are present. Otherwise the algorithm will continue increasing the codebook size, which may not be beneficial.

Vector Quantizer codebooks provide two outputs for each input vector. These outputs are the location number of the closest codeword, and the distortion between the input vector and the codeword. The distortion figure is used for speaker identification. The unknown speaker's identity is decided as the identity of the speaker whose codebook returns the lowest distortion. Figure 5 shows an overview of a vector quantizer based classifier. For a k-dimensional input vector, x, and the nearest codeword, $\hat{x}$, the Euclidean distortion

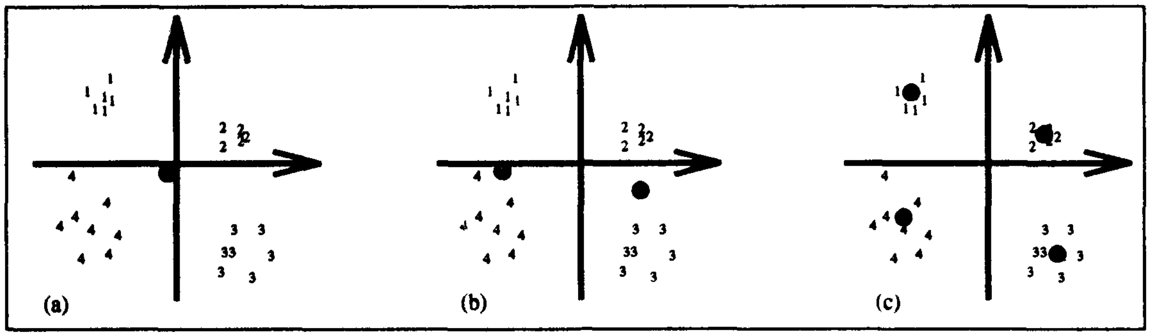


Figure 4. (a) Initial codeword (b) First split (c) Final Codewords

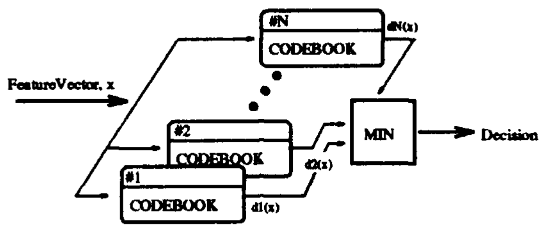


Figure 5. Vector quantizer based classifier

is

$$d(x, \hat{x}) = \|x - \hat{x}\|^2 \qquad (8)$$
$$= \sum_{i=0}^{k-1} (x_i - \hat{x}_i)^2$$

In order to identify a speaker the distortion of the utterance is calculated using each of the known speakers codebooks. The identity is chosen using the lowest distortion.

2.5 Fuzzy Clustering

Fuzzy clustering techniques provide an intuitive and useful tool in pattern recognition. Bezdek [6] provides a very comprehensive discussion of fuzzy algorithms and their application to pattern recognition. Fuzzy mathematics assigns memberships to each data point, one membership for each of the cluster centers in the data space. A sample which is close to a cluster center, or centroid, has a high membership, close to one, in that cluster and much lower memberships in the other clusters. The degree of membership provides an

indication of how typical the sample is in a given cluster. Consider a sample due to a noise process, which is an outlier in a data set. Conventional, or hard, clustering algorithms will assign the point to a cluster even if it is detrimental to do so. In comparison, fuzzy clustering will assign the data point very low memberships in all clusters, indicating that it is not typical of the data set. The membership value u_{ij}, of a sample j, in cluster i is defined in the interval

$$0 \leq u_{ij} \leq 1$$

The membership value is generally based on the inverse of the distance from the cluster center to the sample point. The membership is defined as follows :

$$u_{ij} = \frac{1/d(X_j, V_i)}{\sum_{k=1}^{K} 1/d(X_j, V_k)} \tag{9}$$

Where :

u_{ij} : is the membership of sample X_j in cluster V_i

K : is the total number of clusters.

$d(X_j, V_k)$: is the distance from sample X_j to centroid cluster V_i

Figure 6 illustrates the membership function of a point as a function of normalized distance from a cluster center. The figure shows a typical membership function, however these vary between authors and applications [6, 10, 19, 29].

Fuzzy logic came into prominence with the 1965 paper "Fuzzy Sets" by Lotfi Zadeh [51]. Since then fuzzy theory has been applied to many areas including pattern recognition. Ruspini provided one of the first applications of fuzzy logic in clustering, he extended the conventional k-means algorithm into the Fuzzy k-Means (FKM)[44]. Figure 7 is an example Ruspini used to illustrate the memberships of data points using fuzzy clustering. The top of the figure shows two triangular shaped clusters, commonly known as Ruspini's butterfly. The lower section of the figure shows the memberships in each of the two classes.

x-direction The work by Ruspini was extended by Dunn [13] with the proof of convergence for the Fuzzy c-Means published in 1980 and 1987 [5, 7].

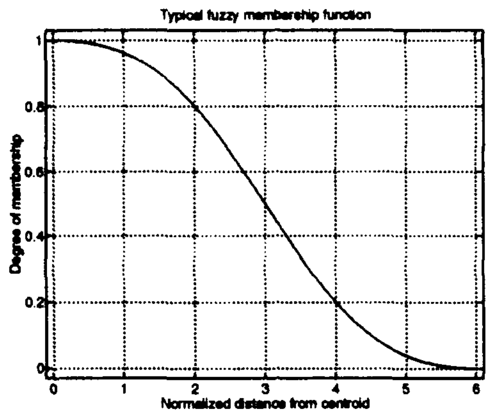


Figure 6. Typical Fuzzy membership function vs Distance from cluster

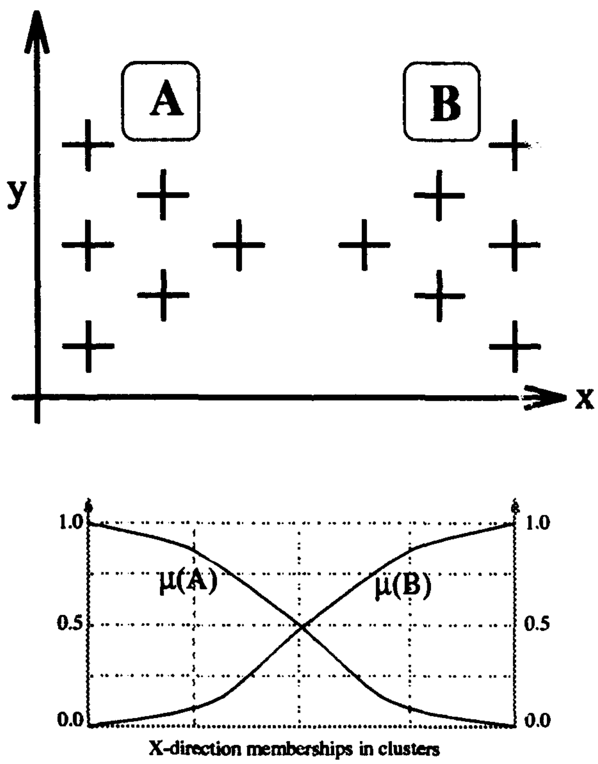


Figure 7. Memberships functions of a data set (after Ruspini [44])

The Fuzzy k-means clustering algorithm is described in pseudo-code below :

1. Select the number of clusters required and initialize their positions randomly,
2. Compute the membership of each data point for all the clusters,
3. Compute new cluster centers using the new membership values,
4. Compute the membership of each data point for all clusters,
5. Is the $\max_{ij}[|\hat{u_{ij}} - u_{ij}|]$ less than the stopping criteria?
   - IF yes, THEN stop.
   - ELSE repeat steps 3 to 5.
6. Save final cluster positions for use in classification.

Where $\max_{ij}[|\hat{u_{ij}} - u_{ij}|]$ is the objective function and $\hat{u}$ is the new membership matrix.

2.5.1 The Fuzzy k-Means Algorithm. The fuzzy k-means algorithms is shown below. The FKM is listed in *C* code and MATLAB script in the Appendix.

1. Initialize the centroids, V_i, using any suitable method,
2. Compute the distance from each centroid to each data sample using

$$d(X_j, V_i) = (X_j - V_i)^T A^{-1}(X_j - V_i) \tag{10}$$

   where A is a positive definite matrix. If A is the identity matrix then the distance measure is the Euclidean distance.

3. Compute the next iteration of centroids $\hat{V}$, for some $q > 0$. The parameter, q, is known as the fuzziness and in this work $q = 2$.

$$\hat{V}_i = \frac{\sum_{j=1}^{N}(u_{ij})^q X_j}{\sum_{j=1}^{N}(u_{ij})^q}$$

4. Save the previous memberships, u_{ij}
5. Update the data memberships, $\hat{u_{ij}}$ using Equation 10

$$\hat{u_{ij}} = \frac{[1/d(X_j, V_i)]^{(q-1)}}{\sum_{k=1}^{K}[1/d(X_j, V_k)]^{(q-1)}}$$

6. Calculate the objective function

$$\max_{ij}[|u_{ij} - \hat{u_{ij}}|]$$

7. If the (objective function $< \epsilon$), then STOP otherwise GOTO step 3 and continue iterating.Where $\epsilon \in [0,1]$.

The *fuzzy* k-means and the *hard* k-means both form spherical clusters around the centroids, this does not always reflect the true shape of clusters. To better exploit cluster shapes Gustafson and Kessel [24] introduced the Fuzzy covariance matrix into the FKM algorithm. This work was extended by Gath and Geva in their Unsupervised Fuzzy Partition-Optimal Number of Clusters (UFP-ONC) algorithm [19]. In their paper Gath and Geva introduce a method for determining the optimal number of cluster present in a data set. Optimality is determined using cluster validity criteria. The cluster validity measures are the fuzzy hyper-volume, partition density and average partition density. Simply put, each cluster is specified by a centroid, a fuzzy-covariance matrix and the *a priori* probability of the cluster. The fuzzy covariance specifies the local geometry of each cluster, this technique is widely used in image processing for the detection of lines for applications such as computer vision and satellite image processing. Cluster validity measures assess each cluster by measuring its volume (in n-dimensions) based on the fuzzy covariance matrix. This can be thought of as building a shell around the cluster, with the shell wall being one standard deviation from the center in each of the dimensions. The smaller this volume the more compact the cluster, which is a desirable outcome. However, in the extreme case where hyper-volume is maximized each data point could be considered a cluster. The benefit of the clustering would be lost, so a second criteria, the cluster density, is introduced to ensure that the clusters are both compact and densely populated.

To determine the optimal number classes the Unsupervised Fuzzy Partition-Optimal Number of Classes (UFP-ONC) algorithm computes the cluster validity criteria starting with two centroids. The number of centroids is then increased to three, again the cluster validity measures, fuzzy hyper-volume, partition density and average partition density are calculated. The number of centroids is progressively increased until the maximum (set by the user) is reached. By examining the cluster validity measures against the number of clusters an optimum may be found. This optimum should provide small fuzzy hyper-volume, high partition density and high average partition density. The FMLE section of the UFP-ONC must be applied carefully, if the UFP-ONC is initialized poorly, the algorithm will not converge to the true cluster centers. This is due to the fact that the distance measure, Equation 11, and the fuzzy covariance matrix, confine the centroids to

a small area, effectively restricting the search of the feature space. To overcome this the UFP-ONC uses the fuzzy k-means algorithm to initialize the centroids and then refines the centroid locations using the fuzzy maximum likelihood estimation algorithm.

The UFP-ONC algorithm in pseudo-code can be summarized as

1. Set the number of clusters at two
2. Repeat until maximum number of clusters computed
   - Compute the centroid locations using the fuzzy k-means
   - Compute the Fuzzy Maximum Likelihood Estimation algorithm
   - Compute the cluster validity measures,
3. Analyze the validity measures to determine the optimal number of classes

2.5.2 The Fuzzy Maximum Likelihood Estimation Algorithm. The fuzzy maximum likelihood estimation (FMLE) [19] algorithm is shown below. The *C* code is included in the Appendix.

1. Initialize the centroids using the fuzzy *k*-means
2. Set the initial data *posterior* (memberships in the FKM) to

$$h(i|X_j) = \frac{1}{K} \text{ where } K \text{ is the number of clusters}$$

3. Set the initial fuzzy covariance to the identity matrix
4. Compute the *a priori* probability of each cluster

$$P_i = \frac{1}{N}\sum_{j=1}^{N} h(i|X_j)$$

5. Compute the *exponential* or fuzzy Mahalanobis distance from each centroid to each data sample using

$$d_e^2(X_j, V_i) = \frac{[det(F_i)]^{\frac{1}{2}}}{P_i} exp\left[(X_j - V_i)^T F_i^{-1}(X_j - V_i)/2\right] \tag{11}$$

where F_i the fuzzy covariance matrix is defined by Equation 12.

6. Compute the next iteration of centroids $\hat{V}$,

$$\hat{V}_i = \frac{\sum_{j=1}^{N} h(i|X_j)X_j}{\sum_{j=1}^{N} h(i|X_j)}$$

7. Save the previous *a posterior* probabilities (memberships),

$$h(i|X_j)$$

8. Update the data memberships, $h(i|X_j)$ using

$$\hat{h}(i|X_j) = \frac{1/d_e^2(X_j, V_i)}{\sum_{k=1}^{K} 1/d_e^2(X_j, V_k)}$$

9. Calculate the objective function

$$\max_{ij}\left[|h(i|Xj) - \hat{h}(i|Xj)|\right]$$

10. If the (objective function $< \epsilon$), then STOP otherwise continue iterating using Equation 12 below .
11. Compute the fuzzy covariance matrix, F_i for each centroid

$$F_i = \frac{\sum_{j=1}^{N} h(i|X_j)(X_j - V_i)(X_j - V_i)^T}{\sum_{j=1}^{N} h(i|X_j)} \tag{12}$$

Fuzzy techniques have been applied in conjunction with vector quantization for use in a speech recognition system. Tseng, Sabin and Lee [48] used a fuzzy vector quantizer (FVQ) to reduce the amount of training data required for a Hidden Markov Model (HMM). In that application an LBG based vector quantizer was modified to output a membership function vector. This combination of LBG and fuzzy memberships provided a technique for reducing the distortion in the output of the vector quantizer. The memberships were used to assist the Hidden Markov model to determine the state transition probabilities. Fuzzy techniques and vector quantization fusion has also been reported by Asakawa et al [2, 3]. This work used the LBG/Fuzzy combinations for improving the fidelity of speech transmitted using low bit rates of 2400bps. These applications of fuzzy techniques illustrate that some benefit is offered by introducing fuzzy methods into speech processing problems.

2.6 Artificial Neural Networks

Artificial Neural Networks comprise a wide variety of different algorithms and architectures. They all share common attributes of highly interconnected nodes and a learning equation. The success of the human neurological system is the basis for artificial neural networks. From the pattern recognition standpoint ANNs are non-linear classifiers. The multilayer perceptron can learn arbitrarily complex decision boundaries, provided enough nodes are used [12]. Rogers and Kabrisky [43] discuss the different types of non-linearities that can be used. Lippmann [31] in his excellent tutorial, provides a extensive discussion of the multilayer perceptron (MLP) and the back-propagation algorithm which *trains* the network.

The back-propagation algorithm is a gradient descent algorithm designed to minimize the mean square error between the actual output of the network and the desired output. The training process is supervised training, this means that the data set is labelled to indicate which class each of the feature vectors represents. Each feature vector is presented to the MLP, each feature is weighted and fed forward to a hidden layer of nodes. Each node sums its input and then applies the non-linear (typically sigmoid) function. The first hidden layer outputs are then weighted and fed to the following layer, this continues until the final output layer is reached. The output vector is compared to a desired output set

by the user. The error is then back-propagated through each layer of nodes. At each layer the weights are adjusted using the back-propagation algorithm. Recent research at AFIT by McCrae, Keller and Martin, using neural networks has been successful. McCrae [34] used an MLP for color image segmentation, as a pre-processor for a target identification system. Keller [27] successfully used neural networks for personnel identification by fusing face data and speech data and Martin [32] applied neural networks to radar identification of non-cooperative targets.

2.7 Conclusion

This chapter provides a review of literature relating to speaker identification, cepstral features, minimum distortion classification and clustering techniques. The review of the cepstral features included both the Fourier Transform method, and the Linear Predictive Coding method of finding the cepstral coefficients. The cepstral features provide information about the vocal tract transfer function of a speaker. By using clustering algorithms we attempt to find a unique set of parameters which uniquely identify that person.

Clustering techniques attempt to partition the feature space into regions defined by a centroid and its neighborhood. Having partitioned a feature space we can construct a classifier which provides the heart of the speaker identification system. The clustering algorithms reviewed are vector quantization, specifically the algorithm by Linde, Buzo and Gray, fuzzy clustering, with emphasis on work by Gath and Geva and artificial neural networks. Vector quantization methods are now well developed and widely used in communication system, in this work they provide a baseline for evaluating fuzzy clustering and ANN performance. Fuzzy clustering offers a technique for dealing with data points which are near the boundary of two partitions. These points often present difficulties for clustering algorithms as they may be from either cluster. Fuzzy clustering attempts to alleviate this problem by assigning memberships, the higher the membership the more likely that the point is in a given cluster. Artificial neural networks provide a dramatically different approach to clustering, the network is able to construct arbitrarily complex decision boundaries to partition the feature space. ANN's use relatively simple functions such as sigmoids inside a highly interconnected network. All three methods attempt to partition

the feature space through a *learning* process. These techniques are extremely useful when the process is too complex to be deterministically modelled.

Chapter III discusses the structure of the data sets, including the use of separability measures, and finally the implementation of the clustering algorithms.

III. Approach and Methodology

3.1 Introduction

Three different clustering methods were used in this speaker identification experiment, vector quantization, fuzzy clustering and a multi-layer perceptron. The following sections describe the each of the data sets, the speech capture and preprocessing, and the implementation of each of the clustering algorithms.

3.2 Data Sets

Three data sets were used in this work, the TIMIT and King speech databases and a local AFIT corpus which was recorded during this research. The AFIT93 corpus comprises twelve speakers, eleven male and one female. Ten of the speakers were recorded in ten sessions over a three week period. The remaining two were recorded for seven sessions over the same three week period. Each session comprises the person's full name, uttered three times, followed by three sentences based on the phonetically balanced sentences of the TIMIT database.

A portion of the DARPA TIMIT Acoustic Phonetic Continuous Speech Database [36] was used to provide a corpus of ten speakers, seven male and three female, each speaking ten sentences. The TIMIT database provides good quality recordings, with a signal to noise ratio of 36.72dB. The TIMIT sentences used were the

- SA
- SX
- SI

Examples of the phonetically balanced TIMIT phrases are included in the Appendix.

The King database was collected by ITT Aerospace and comprises two recording methods,wide and narrow band. The narrowband recordings of the first twelve speakers were used in this work. Each speaker is recorded in ten sessions, with each session 60 seconds in length. The King database was collected in two stages using different equipment, the first five sessions are of much higher quality than the second five. Colombi [11] provides

details of the signal to noise ratios with and without silence frames, the maximum is 14.75dB, significantly lower than TIMIT.

The AFIT93 recordings were made in a computer room, using an Ariel Proport and SUN Sparc 2 station. The speech was recorded at 16KHz and later down sampled to 8KHz in keeping with the sampling rate of King and TIMIT.

3.3 Data Pre-processing and Feature Extraction

Each of the data sets was processed using the Entropic Signal Processing System [14] which is a commercial software package comprising a library of programs for speech processing. The data was pre-emphasized with a $1 - 0.97z^{-1}$ filter, followed by a Hamming window using 256 samples (32ms) per frame, with consecutive frames overlapping by 85 samples. The next stage of processing involves two processes, LPC-cepstral coefficient extraction and calculation of formants. The formant information is used to calculate the probability of voicing of the frame of speech. The probability of voicing is then appended to each cepstral feature vector and used to select only feature vectors which have greater than 10% probability of voicing. These feature vectors are then used in the clustering experiments. Figure 8 provides an overview of the pre-processing system. The following code is *C* script file [21] containing the ESPS commands which implement the data preprocessing.

```
#!/bin/csh
#
# This script uses ESPS to extract cepstral features from the TIMIT corpus
# database collected by Texas Instruments
#
foreach FILE (*.sd)
                filter -f filter -P preemp_params transit.sd transit.flt.sd
                refcof -P ../Prefcof transit.flt.sd transit.ref.cof
                spectrans -m"CEP" transit.ref.cof transit.cep
                formant -p.97 -w.032 -W1 -i.0106 -F transit.sd
                pplain -e2 transit.f0 > pv
                addfea -f prob_voice -c"Adding prob_voice" pv transit.cep
                set TEMP = $FILE:r
                select -q "prob_voice > .1" -o $TEMP.cep transit.cep
end
```

The final pre-processing step was to partition the databases into equal halves for training and testing of the classifiers.

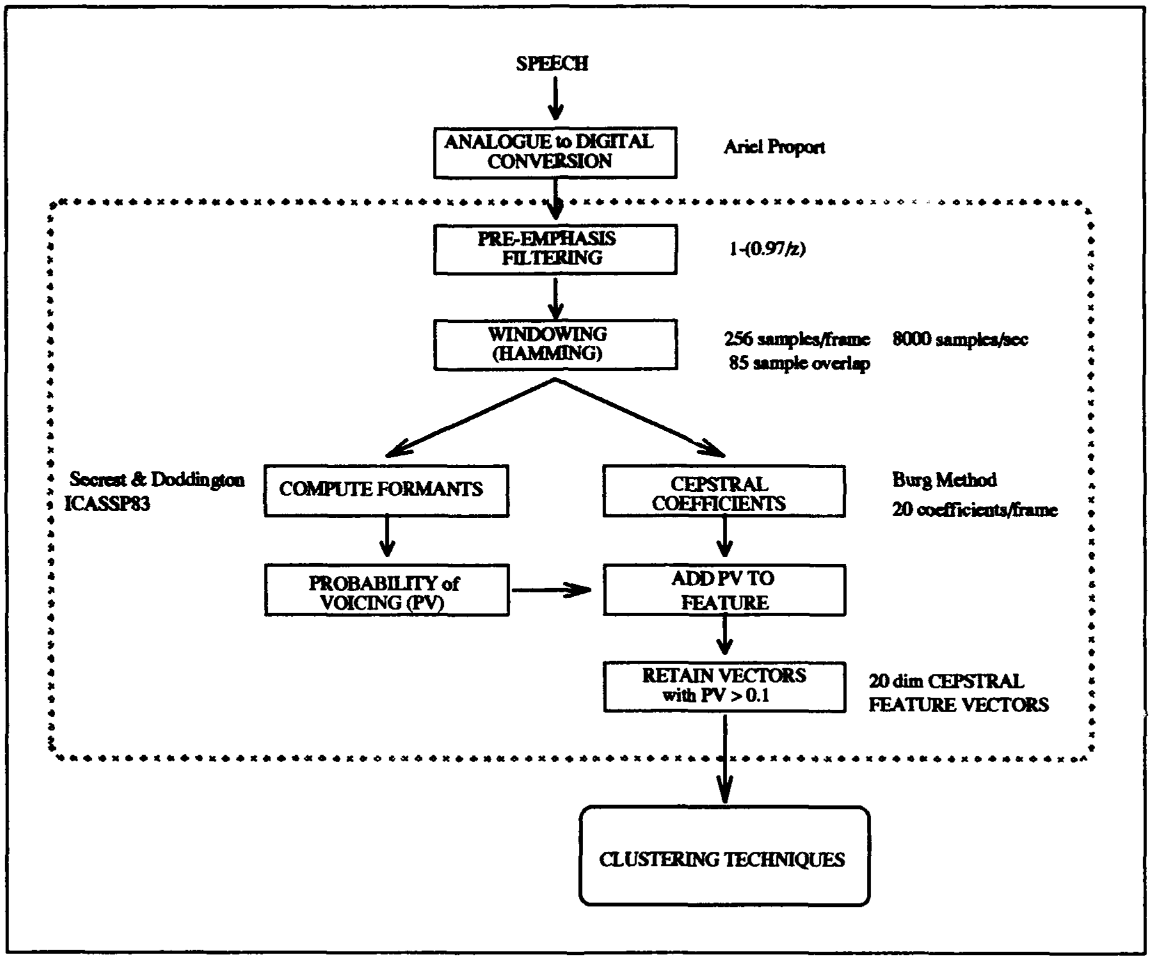


Figure 8. Cepstral Feature Extraction Process

Table 1. Data Selection used for Training and Testing

Test No.	Sessions used in Training	Sessions used in testing
1	1	1, 2, 3, 4, 5 (6, 7)*
2	1,2	1, 2, 3, 4, 5 (6, 7)*
3	1,2,3	1, 2, 3, 4, 5 (6, 7)*
4	1,2,3,4	1, 2, 3, 4, 5 (6, 7)*
5	1,2,3,4,5	1, 2, 3, 4, 5 (6, 7)*
6*	1,2,3,4,5,6	1, 2, 3, 4, 5, 6, 7
7*	1,2,3,4,5,6,7	1, 2, 3, 4, 5, 6, 7

* AFIT93 only

The first seven days of the AFIT93 database was split into training and testing sets. This provided seven sessions of data with each session comprising a training and testing set.

All ten of the King sessions, for speakers one to speaker twelve, was divided into equal training and testing sets.

TIMIT speakers fcmm0, fcrh0, fedw0, mcmj0, mefg0, mhpg0, mjls0, mmwh0, mprk0, mrtk0, were used in the experiment. The utterances from each speaker were divided into training and testing sets in the same manner as for King and AFIT93.

The identification rate was recorded for classifiers trained using an increasing number of sessions. Based on the assumption that codebooks designed with data from a number of sessions would better represent the long term statistics of the speakers cepstral features, it is expected that the identification rate should increase with the number of training sessions present. Table 1 shows which training/testing data was used for each of the experiments.

3.4 Vector Quantization

The vector quantization speaker identification system was developed using the ESPS libraries in an identical method to that used in previous work at AFIT by John Colombi [11]. The training data for each database was used to generate a vector quantizer codebook for each speaker using the **vqdes** (LBG) algorithm available in ESPS. These codebooks were then combined into a global codebook which was used as the classifier.

The identification process presents the feature vectors of an utterance to the global codebook, the utterance is then compared against each of the speaker codebooks. For each codebook the distortion is computed using the Euclidean distance of the feature vectors to each of the codewords. The classifier then chooses the speaker codebook which has the lowest distortion.

Define $D(\omega_i)$, the utterance distortion

$$D(\omega_i) = \sum_{k=1}^{K} \min_j \|u_k - c_{ij}\| \tag{13}$$

we choose class ω_1 if

$$D(\omega_i) = \min_{i'} D(\omega_{i'})$$

where:
- u_k : feature vector to be classified
- c_{ij} : codeword j from speaker i
- K : is the number of feature vectors in the utterance.
- j : is the number of codewords in each codebook, and
- i : is the number of known speakers.

In this experiment, the codebooks have 64 codewords for each speaker, so for example the AFIT93 VQ classifier decision is given by

$$D(\omega_i) = \min_{i=1\cdots 12} \sum_{k=1}^{K} \min_{j=1\cdots 64} \|u_k - c_{ij}\|$$

3.5 Fuzzy Clustering

The identical training data as used in the VQ classifier was clustered to produce a set of centroids for each session, for each speaker. The Fuzzy clustering algorithm was written by the author using ANSI *C* [28] and includes routines from "Numerical Recipes in C" [41]. The fuzzy clustering algorithm is performed in two distinct steps. The initial clustering is performed using the Fuzzy *k*-means (FKM). After the FKM has converged the second phase introduces the Fuzzy Maximum Likelihood Estimation (FMLE) algorithm published by Gath and Geva. The FMLE algorithm computes a covariance matrix for each

of the centroids. This is then used in the calculation of the distances of data points from the centroid. The distances are subsequently used for membership calculations. The UFP-ONC algorithm is discussed in detail in chapter II.

A number of issues were encountered during the application of the UFP-ONC to speaker identification. The calculation of the inverse of fuzzy covariance matrices posed significant difficulties that were not anticipated. The fuzzy covariance matrix was often singular or near singular, which prevents finding the inverse. Initially the inverse was calculated using LU decomposition, however this was replaced with Singular Value Decomposition (SVD). SVD provides a means of detecting singularity and was used to compute the pseudo-inverse of the covariance matrix when the complete inverse could not be computed. There appear to be a number of reasons for the covariance matrix becoming singular. The most obvious case is where points are collinear or coplanar. The most likely source of problems is the *exponential* (Equation 11) distance measure. The distances can become extremely large and cause the memberships become very small, often zero. This, in turn, is reflected in the calculation of the fuzzy covariance matrix, which may lead to it becoming singular even though the members of the cluster are not coplanar.

Another issue which was encountered is the problem of repeated centroids. This occurs when two, or more, centroids are located at exactly the same point. The repeated centroid problem is mainly attributed to the fuzzy k-means algorithm which is used to make the initial estimate of the centroids. This problem is not confined to fuzzy algorithms. Bezdek [6] discusses these types of problems and a number of techniques to prevent this. A solution to this problem was not found, although it is an important addition that the UFP-ONC requires to ensure its correct operation.

These issues, which the author considers to be the fragility of the UFP-ONC, require additional research.

The programs were run using SUN Sparc 2, Sparc 10, IBM RS6000 and Silicon Graphics Iris 4D computers. A full listing of the code is included in the appendix. The *C* program reads a configuration file which details the parameters to be used for a given clustering run. An example of the configuration file is also included in the appendix.

3.6 *Fuzzy Classifier*

The final component of the speaker identification is the classifier. Gath and Geva achieved classification by examining the membership values after the clustering process had converged. A few comments about the fuzzy maximum likelihood estimation algorithm are required as it has considerable impact on the design of the classifier.

The FMLE uses fuzzy covariance matrices to define local neighborhoods around each centroid, this combined with the distance measure produces memberships. In this author's experience, these memberships are very close to one, or very close to zero. When the entire data set is clustered at one time these memberships provide an excellent classification as Gath and Geva [19] demonstrated with Anderson's [1] Iris data.

The speaker identification experiments conducted in this work treated each speaker separately. The membership values indicate which cluster of speaker X a feature vector should be assigned to. However to identify a speaker requires a method of determining which speaker is the closest using some form of distance metric. To achieve this type of distortion based classification, a rudimentary classifier was designed based on the Maximum Likelihood classifiers discussed in Tou and Gonzalez [47:Sect 4.4]. It is effectively the same process as the LBG classifier shown in Equation 13. The fuzzy maximum likelihood classification algorithm proceeds as follows

1. Present an unknown utterance
2. For each known speaker
   - Load centroids and Fuzzy covariance matrices
   - Compute the Fuzzy Mahalanobis distances using Equation 11 for all feature vectors
   - Determine the distance from the centroid with the highest membership
   - Total the log of these closest distances
3. Find the minimum total distance, classify the utterance to be from that speaker

The classifier was written in *ANSI C* and is included in the appendix. This classifier is a ***"quick and dirty"*** approach which needs closer examination and some refinement. However, the classification system performed quite well.

3.7 Artificial Neural Network Multi-layer Perceptron

AFIT has conducted considerable research using artificial neural networks. The Multi-layer perceptron code used in this research was written by Curtis Martin for his masters thesis [32]. The neural network code is written in ANSI *C* and was executed on the same computers noted in the Section 3.5 on fuzzy clustering. The code has been modified by this author to include the ability to save and load weights and to remove the multiple testing mode.

There are no hard and fast methods to determine how many hidden nodes, should be used in a neural network. The basic rules are too few and the error rate will be high, too many and the network will memorize the training data and not generalize. Hush and Horne [26] discuss this issue at some length. The method used in this research is based on Widrow's heuristic :

$$10 * [(m+1) * H_n + (H_n + 1) * C_o] < K \tag{14}$$

Where
m : is the dimension of the feature vectors
H_n : is the number of hidden nodes
C_o : is the number of output nodes (classes)
K : is the total number of data training samples

For example consider designing an MLP for the following classification parameters,

- $m = 20$ dimensional speech features
- $C_o = 12$ speakers
- $K = 20,000$ sample feature vectors

$$10 * [(20+1) * H_n + (H_n + 1) * 12] \quad < \quad 20,000$$

$$\Rightarrow 33 * H_n + 12 < 2000$$
$$\Rightarrow H_n < (2000 - 12)/33$$
$$\Rightarrow H_n < 60.24$$

3.8 Data Separability

Pattern recognition is the process of identifying all the different classes in a population, based on the features used to represent the population.

Consider the data points in Figure 9 below, there are three classes of artificially generated gaussian distributed data. The (x, y) coordinate pairs provide a feature set which can be used to separate the data into the three classes. Using this feature set we quickly determine that two classes overlap, while the third class is clearly separated from the other two.

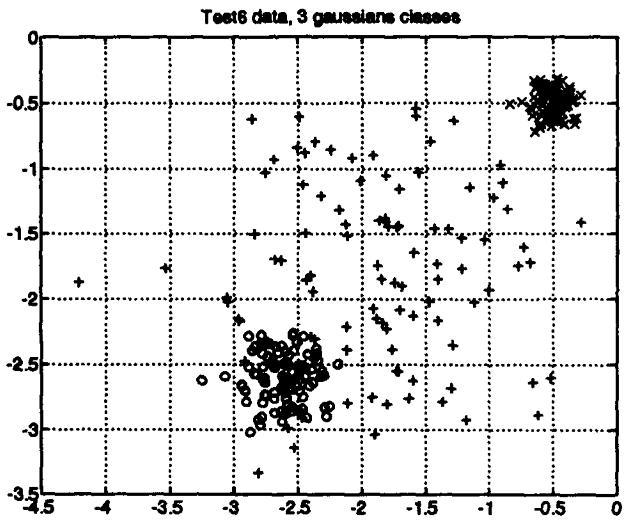


Figure 9. Test6 data

When the dimension of the feature vectors is increased beyond three we humans can no longer provide a simple graphical representation, and the intuitive feel for class separability is lost. The feature vectors used in this work are 20-dimensional cepstral coefficients, with the number of classes equal to the number of speakers, twelve for AFIT93

and King, and ten for TIMIT. Estimating the degree of separability of the classes indicates how difficult the class identification task is. Additionally, separability measures provide a guide to which pattern recognition algorithms are likely to be successful. This aspect of data pre-processing is relevant to any pattern recognition task, such as image segmentation of synthetic aperture radar data, automatic target identification systems and automated hand written character recognition. This remainder of this section describes a separability measure developed by Fukunaga [17], then applies the separability measure to two simple data sets, Test6 and Anderson's Iris data [1]. The *J4* separability measures for the speech databases is presented in Chapter IV.

The Test6 set is three classes of 2-dimensional gaussian distributed data, each class containing 100 samples. Anderson's [1] Iris data is 4-dimensional and represents the sepal length, sepal width, petal length and petal width of three families of Iris. There are 50 feature vectors from each of the Iris families, Iris Sestosa, Iris Versicolor and Iris Virginica. This data was used by Fisher in his development of linear discriminant analysis [15].

Fukunaga develops a number of separability measures of his book, "Introduction to Statistical Pattern Recognition",[17]. The measures labelled *J1, J2, J3* and *J4* [17:Sect 9.2], provide different indicators of *between-class* scatter as compared to *within-class* scatter. Parsons [40:pp 176-180] provides an introductory discussion on separability measures including Fisher Ratios and Fukunaga's *J1* $\cdots$ *J4* indicators.

The measure *J4* by Fukunaga [17] is defined by

$$J4 = \frac{trS1}{trS2}$$

where, $trS1$ and $trS2$, are the trace of inter-class scatter matrix, and the intra-class scatter matrix, respectively. The trace of a matrix is defined [46] as

$$trA = \sum_{i=1}^{k} a_{ii} = \sum_{i=1}^{k} \lambda_i$$

where λ_is are the eigenvalues of the matrix A, which is kxk. The eigenvalues represent the major and minor axes of an k-dimensional hyper-ellipsoid, (where $\lambda_i > 0$) in this case the

Table 2. Separability Measure $J4$

Iris	Test6
19.849830	13.225790

cluster of feature vectors representing a class. The scatter matrices, $S1$ and $S2$ are defined by

$$S1 = \sum_{i=1}^{M} P(\omega_i)(M_i - M_o)(M_i - M_o)^T \tag{15}$$

$$S2 = \sum_{i=1}^{M} P(\omega_i)E\left\{(X - M_i)(X - M_i)^T/\omega_i\right\} \tag{16}$$

Where M is the number of classes in the data population. Equation 16 is often represented as

$$S1 = \sum_{i=1}^{M} P(\omega_i)C_i \tag{17}$$

$$C_i = \frac{1}{N}\sum_{i=1}^{N}(X_i - M_j)(X_i - M_j)^T \tag{18}$$

where:
- $P(\omega_i)$: is the a priori probability of class i
- M_i: is the mean of class i
- M_o: is the global mean of the whole data set
- C_i: is the covariance matrix of a class i.
- N : is the number of feature vectors in the class

The larger the $J4$ separability measure the more separable a data set is. The break even point is when $J4 = 1$, which means that the inter-class scatter is identical to the intra-class scatter. This would still present a non-trivial pattern classification task.

The separability measures results for the two data sets are shown in Table 2. At first inspection both the Test6 and Iris data appear to be easily separated into three classes.

Separability measures must be considered carefully, as indicators can be misleading. The Iris data is plotted in Figure 10 using three of the four dimensions. Note that two of the classes overlap. This is also the case for the Test6 data in Figure 9. Computing the $J4$ measure for the Iris data set when considering two classes at a time reveals the true

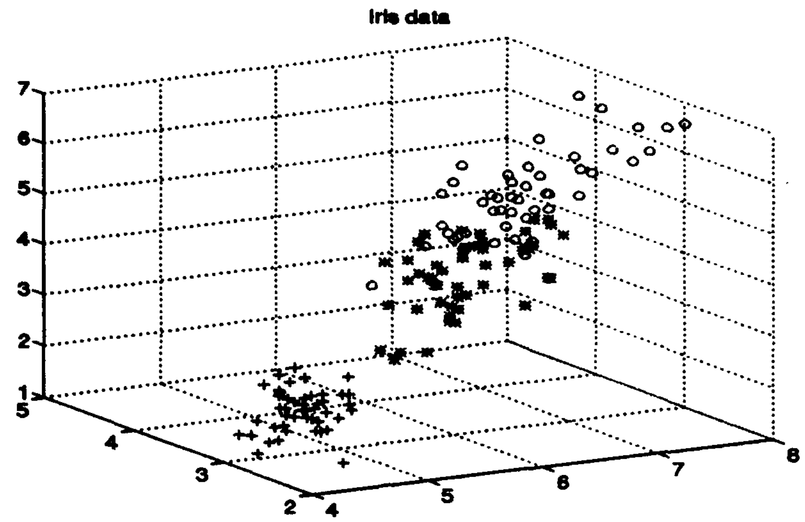


Figure 10. Iris data dimensions 2,3,4

Table 3. Separability measure $J4$ for Iris and Test6

Iris $J4$				Test6 $J4$			
ω_1	ω_2	$\omega 3$		ω_1	ω_2	$\omega 3$	
1	5.601	9.608	ω_1	1	1.842	0.598	ω_1
·	1	0.885	ω_2	·	1	50.105	ω_2
·	·	1	ω_3	·	·	1	ω_3

structure of the data set. This is shown in table 3, the first row shows that class one (Iris Sestosa) is clearly separable from the other two classes (Iris Versicolor and Iris Virginica). The second row reveals the overlap we saw (Figure 10) between the other two classes. The overall $J4$ measure provides a general indicator of the data set, however it is important to consider the classes in pairs to obtain an accurate indication.

3.9 Conclusion

Three speech databases are used in this speaker identification research. The AFIT93, TIMIT and King data were all pre-processed identically using the ESPS speech processing library. The cepstral feature vectors from segments of speech which had greater than ten percent probability of voicing were retained as the feature set. The data was partitioned

into a training and a test set for each day. The identification experiments for each method, vector quantization, fuzzy clustering and neural network, were trained using the data from day one up to the final day. The testing was conducted using the test data from all days. For example the day three experiment was trained using data from day one, two and three. It was then tested using data from all days, five for TIMIT and King and seven for AFIT93. The performance is based on identification rate since the three clustering methods do not have a common distance or distortion metric. Chapter four presents the results of the experiments conducted, including an analysis of the separability, using a separability measure developed by Fukunaga [17].

IV. Results

4.1 Introduction

This chapter presents the results of the speaker-identification experiments. An initial discussion on the separability measures of the databases is followed by the results for each of the clustering techniques.

4.2 Separability Results for Speaker Data

Separability measures were discussed in Chapter III and are used to indicate how difficult the classes are to separate. Table 4 lists the $J4$ separability measure for each of the speech databases. The table shows the separability measure for each speaker when compared to each of the others speakers in turn. Since the matrix is symmetric only the upper half is shown. Recall that the $J4$ measure is the ratio of the inter-class scatter to the intra-class scatter, and for linear separability $J4$ must be greater than one. All three databases present a considerable challenge, King in particular has a large number of classes that appear to be inseparable, at least to three decimal places. Keeping in mind that one is the break even point for the $J4$ measure, displaying three decimal places is only done to ensure that the tables were not filled with zero! It is tempting to conclude that the task of identifying the speakers in these databases is impossible and to stop there. Actually this data provides a justification for using clustering algorithms to design a speaker identification system. Clustering algorithms attempt to detect structure in data sets which is difficult to discern. This is basically sub-partitioning the feature space into smaller regions which (hopefully) provide the separability between classes that we require.

The $J4$ values still provide useful information despite the low values. The relative values indicate which classes are likely to be the most difficult to isolate. The confusion matrices in Section 4.6 confirm the $J4$ values for the AFIT93 speakers. The first speaker, *cm*, is incorrectly identified as being the speaker for the majority of the other AFIT speakers. In the confusion matrix this is indicated by the values in the first column. The $J4$ measures in Table 4 confirm this, since the relative values for *cm* are some of the lowest.

Table 4. Separability Measure $J4$ for AFIT, King and TIMIT

cm	dp	ei	gs	jc	jk	jm	jt	km	mc	rm	wg	
1	0.009	0.006	0.004	0.007	0.007	0.008	0.014	0.010	0.008	0.005	0.018	cm
·	1	0.012	0.011	0.013	0.011	0.004	0.032	0.006	0.005	0.016	0.021	dp
·	·	1	0.002	0.009	0.005	0.010	0.015	0.010	0.016	0.003	0.010	ei
·	·	·	1	0.005	0.008	0.009	0.011	0.008	0.011	0.004	0.017	gs
·	·	·	·	1	0.015	0.013	0.009	0.009	0.017	0.010	0.028	jc
·	·	·	·	·	1	0.006	0.025	0.011	0.014	0.005	0.005	jk
·	·	·	·	·	·	1	0.031	0.007	0.005	0.011	0.017	jm
·	·	·	·	·	·	·	1	0.024	0.035	0.017	0.035	jt
·	·	·	·	·	·	·	·	1	0.012	0.013	0.018	km
·	·	·	·	·	·	·	·	·	1	0.019	0.030	mc
·	·	·	·	·	·	·	·	·	·	1	0.010	rm
·	·	·	·	·	·	·	·	·	·	·	1	wg

sp1	sp2	sp3	sp4	sp5	sp6	sp7	sp8	sp9	sp10	sp11	sp12	
1	0.016	0.039	0.006	0.116	0.045	0.006	0.023	0.001	0.018	0.000	0.000	sp1
·	1	0.009	0.021	0.045	0.011	0.007	0.007	0.015	0.003	0.000	0.000	sp2
·	·	1	0.050	0.022	0.002	0.021	0.003	0.038	0.005	0.000	0.000	sp3
·	·	·	1	0.124	0.060	0.008	0.037	0.003	0.030	0.000	0.000	sp4
·	·	·	·	1	0.020	0.079	0.043	0.111	0.042	0.000	0.000	sp5
·	·	·	·	·	1	0.027	0.005	0.045	0.006	0.000	0.000	sp6
·	·	·	·	·	·	1	0.012	0.005	0.010	0.000	0.000	sp7
·	·	·	·	·	·	·	1	0.024	0.002	0.000	0.000	sp8
·	·	·	·	·	·	·	·	1	0.018	0.000	0.000	sp9
·	·	·	·	·	·	·	·	·	1	0.000	0.000	sp10
·	·	·	·	·	·	·	·	·	·	1	0.000	sp11
·	·	·	·	·	·	·	·	·	·	·	1	sp12

fcmm0	fcrh0	fedw0	mcmj0	mefg0	mhpg0	mjls0	mmwh0	mprk0	mrtk0	
1	0.077	0.164	0.016	0.007	0.008	0.024	0.147	0.063	0.046	fcmm0
·	1	0.016	0.043	0.111	0.104	0.131	0.022	0.008	0.181	fcrh0
·	·	1	0.107	0.223	0.213	0.257	0.010	0.031	0.335	fedw0
·	·	·	1	0.023	0.020	0.038	0.106	0.031	0.064	mcmj0
·	·	·	·	1	0.001	0.013	0.208	0.092	0.021	mefg0
·	·	·	·	·	1	0.007	0.202	0.088	0.020	mhpg0
·	·	·	·	·	·	1	0.251	0.119	0.022	mjls0
·	·	·	·	·	·	·	1	0.028	0.323	mmwh0
·	·	·	·	·	·	·	·	1	0.151	mprk0
·	·	·	·	·	·	·	·	·	1	mrtk0

4.3 Vector Quantization Benchmark

The vector quantization system provides the benchmark performance for comparison with the other techniques. Figure 11 displays the identification rates for the three databases. For all databases the identification rates progressively climb with the number of sessions used in the codebooks. The lower identification rates for the King database are due several factors including dramatic changes in the recording after session five, and the speech is conversational, unlike the phonetically balanced sentences used in TIMIT and AFIT93. The codebooks have 64 codewords and use the Euclidean distortion measure. The performance of the classifier is lower when the test data is used, this drop in performance provides an indication of how well the classifier can generalize.

4.4 Fuzzy Clustering Experiment

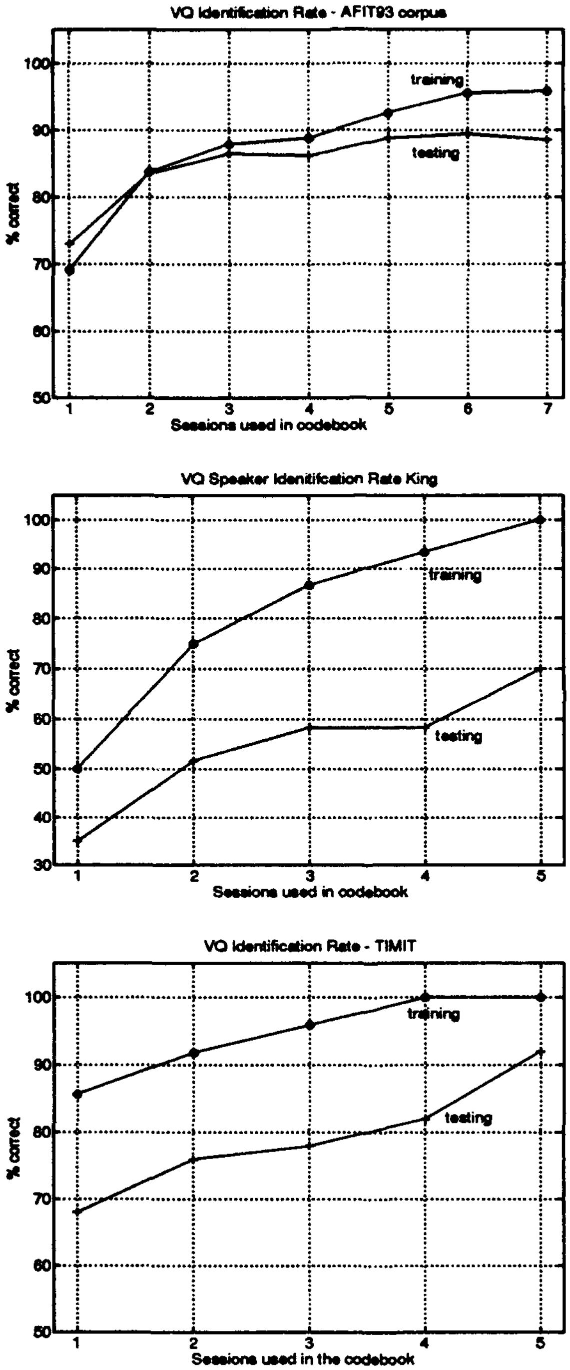
The UFP-ONC based speaker-identification system reported in these results used eight centroids. Figure 12 shows the overall identification rate for the test data sets. The UFP-ONC rates for AFIT93 and TIMIT are 72% and 67% respectively. The result for King is very low, this is most likely due to the UFP-ONC algorithm not converging for speakers sp4 and sp12 on session five. This implementation of the UFP-ONC required convergence in 500 iterations, if convergence is not achieved then the centroids are saved and used as the best set available. This is not an an adequate long term solution, however it is considered to be a reasonable approach.

Figure 13 shows the performance of UFP-ONC, LBG and the MLP for the AFIT93 corpus. The graphs display the mean identification rate with the line. The error bars are plus and minus one standard deviation from the mean. Note that LBG achieves a mean rate of 94.2% while the UFP-ONC achieves 92%, this indicates that the UFP-ONC can achieve performance as high as vector quantization. The vector quantization system is more consistent as seen by the smaller standard deviation. Like the LBG algorithm, the UFP-ONC exhibits better performance as the number of sessions of training data is increased. This appears to confirm that the long term statistics of the speech feature vectors are important to identification accuracy.

4.5 ANN Experiments

Figure 14 displays the identification rate for the multi-layer perceptron. The identification rates are significantly lower than those achieved by vector quantization. Each network was presented with feature vectors from all speakers and trained with one output for each speaker. Had the cepstral features been grouped by families of vowels or phonemes such as the work reported by Rabiner and Juang [42] the identification rates would be higher.

All networks were trained with 20 nodes in the hidden layer, for 3000 epochs. None of the ANNs achieved the desired mean squared error of 0.01, generally the MSE was 0.81 after 3000 epochs. The TIMIT networks were trained for 30,000 epochs but still did not reduce the mean squared error below 0.7. To provide an even comparison between the clustering algorithms the data was presented in exactly the same way each time. This presentation of cepstral data is obviously not suited for classification by MLP. Another factor that must be considered is the training time required for neural networks. The King and AFIT93 databases required four days of processing for 3000 epochs on both the IBM RS6000 and the Silicon Graphics Iris 4D. This made it extremely difficult to conduct multiple tests which may have provided better results.

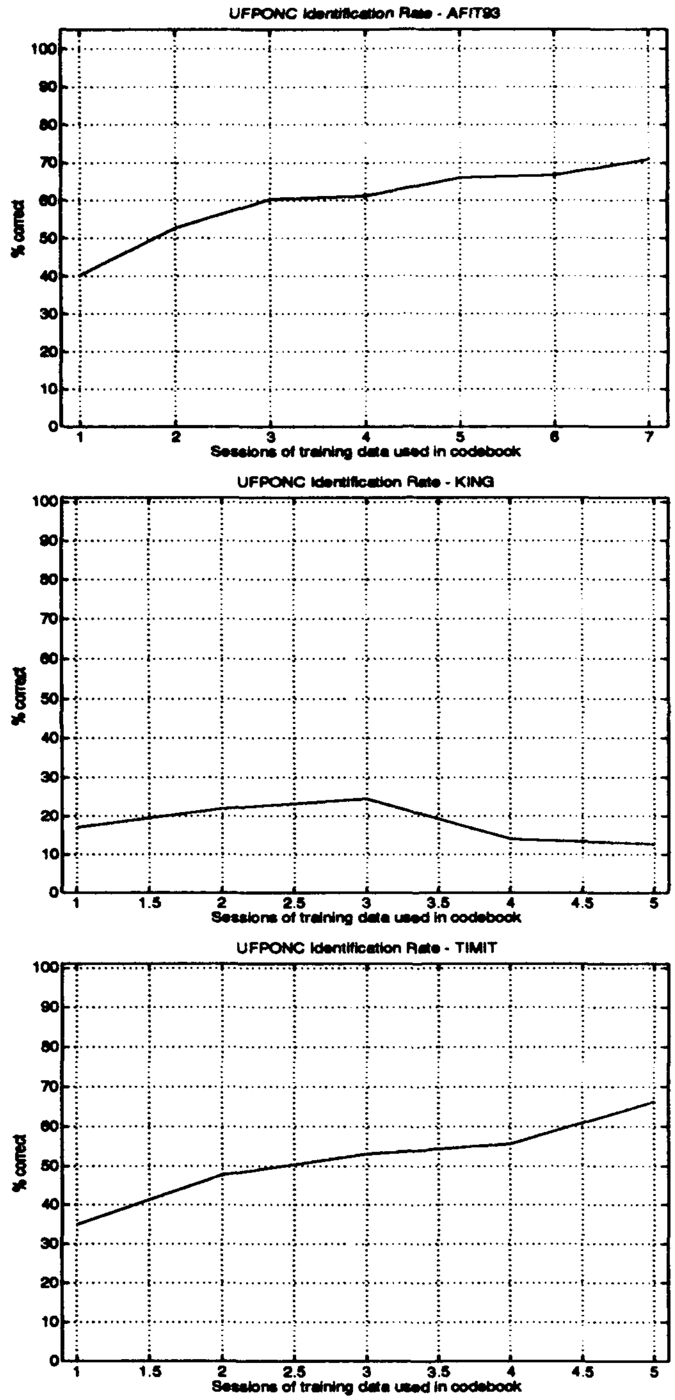


AFIT93

King

TIMIT

Figure 11. Vector Quantizer Identification Rate - Training



AFIT93

King

TIMIT

Figure 12. UFP-ONC Identification Rate

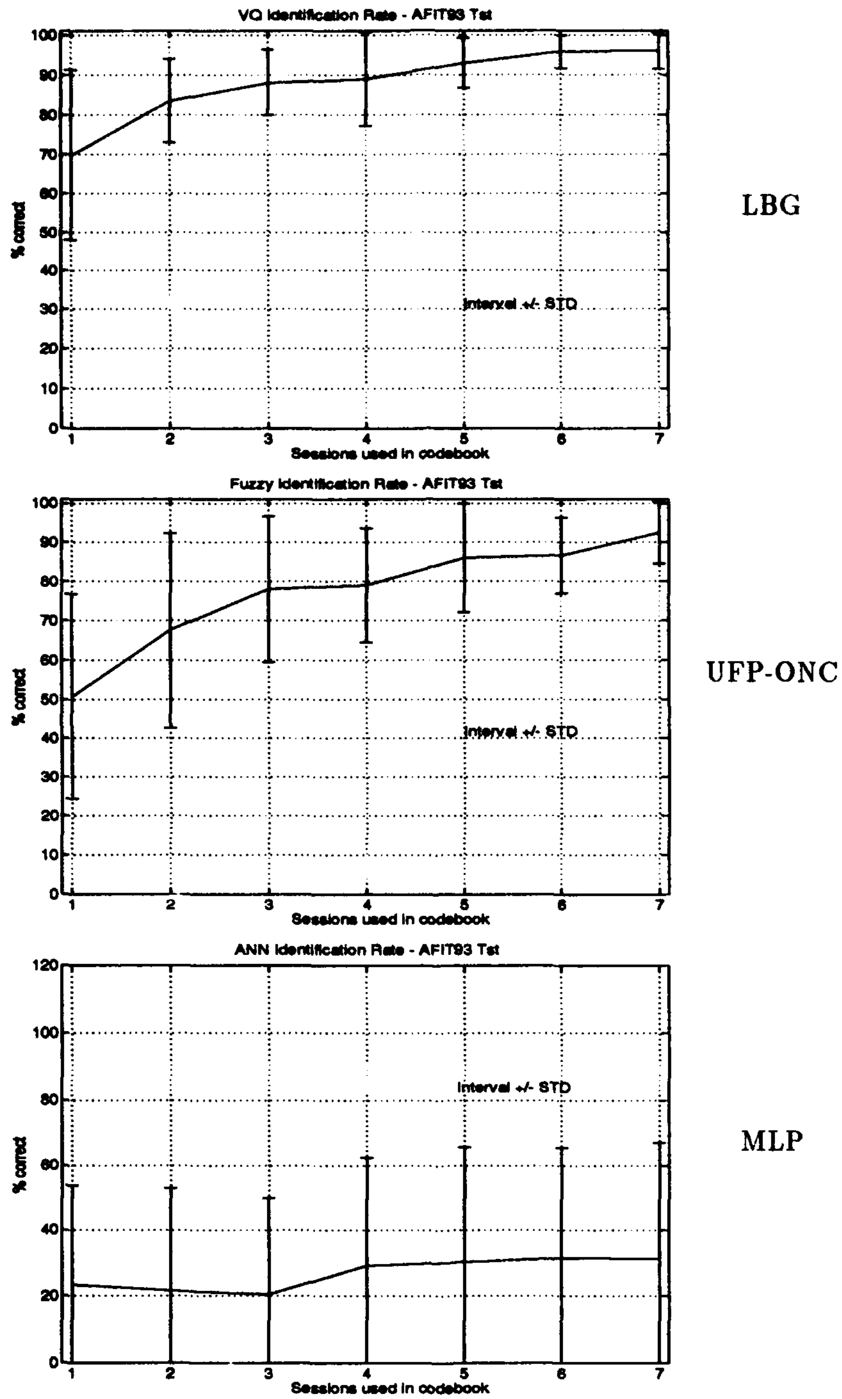


Figure 13. AFIT Identification Rate

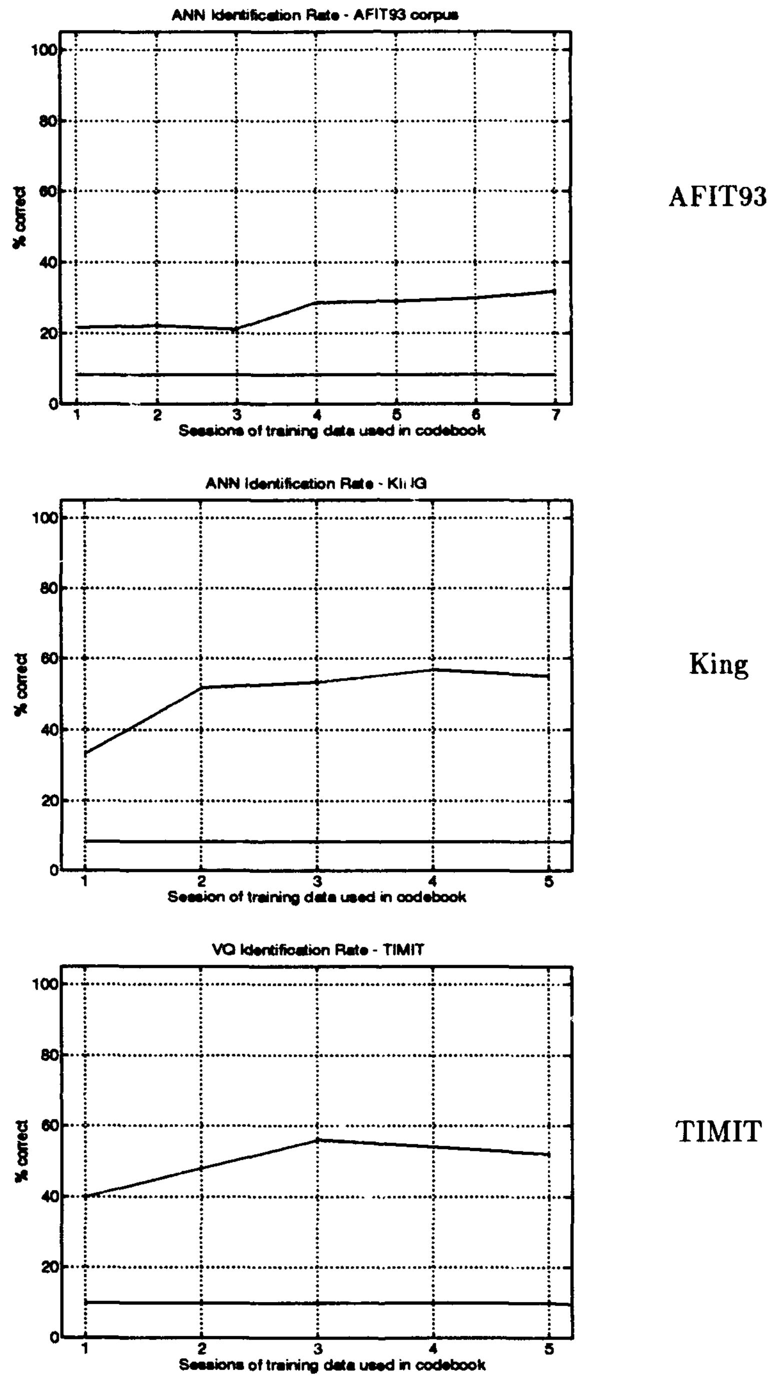


Figure 14. ANN Identification Rate

4.6 *Confusion Matrices*

Confusion matrices for the AFIT93 corpus were generated from the results of the day one and the day seven codebook tests. The confusion matrices are shown in Tables 5 and 6. The vertical axis of the matrix represents the true identity of the speaker, while the horizontal axis represents the decision made by the classifier. For example, looking at Table 5, the second row of the LBG matrix shows that speaker 2 was identified six times as speaker 1, twenty-three times as speaker 2 (correct), and once each as speaker 3 and speaker 5. In addition to showing that the correct identification rate for speaker 2 was 74.2% ($\frac{23}{31}$) the confusion matrix indicates which person was selected when an incorrect decision was made.

The multilayer perceptron appears to have split the speakers into two classes. It appears that attempting to train a multilayer perceptron using all feature vectors from all speakers is not a suitable method. By choosing a subset of speech features, such as the formants, ANNs has been shown to perform extremely well [42]

The UFP-ONC achieves comparable performance to vector quantization. Note that both methods show strong diagonals in the matrix, the results for day seven are especially pronounced. The confusion matrices provide an easy method of determining whether the LBG and UFP-ONC could be used to complement each other, however the confusion matrices also indicate that the vector quantization and the fuzzy technique make similar errors.

4.7 *Conclusion*

This chapter introduced separability measures and provided an analysis of the separability measure for each of the speech databases using Fukunaga's $J4$ measure. The separability measures indicate that all three databases pose a significant problem for speaker identification using any of the statistical based classifiers. Clustering techniques are considered to the most suitable choice to form the basis of a speaker identification system. The identification rates for each of the three techniques was reported, with both the LBG vector quantization and the fuzzy UFP-ONC achieving rates of more than 90% on indi-

Table 5. AFIT Day One - Confusion Matrices for LBG, UFP-ONC, MLP

LBG											
30	.	.	.	.	.	.	.	.	.	.	.
6	23	1	.	1	.	.	.	.	.	.	.
2	.	13	1	3	.	1	.	.	.	.	.
4	.	12	11	1	.	.	1	.	1	.	.
1	.	.	.	27	.	.	1	.	1	.	.
1	.	.	.	.	27	.	.	.	2	.	1
1	.	4	.	.	.	18	.	1	6	.	.
1	.	1	.	3	.	.	24	.	.	.	1
1	2	.	.	.	1	.	.	24	1	.	.
.	1	.	.	.	.	.	.	.	28	.	.
15	1	3	.	.	1	.	1	.	2	7	.
3	.	.	.	.	1	.	.	.	.	.	17

UFP-ONC											
29	.	.	.	.	.	.	.	1	.	.	.
4	24	.	.	.	.	.	.	3	.	.	.
1	.	2	10	.	1	.	.	.	4	2	.
1	.	.	28	.	.	.	.	.	1	.	.
9	.	.	5	6	.	.	.	.	5	5	.
2	.	.	.	.	20	.	.	.	8	1	.
.	.	1	1	.	.	.	.	.	25	3	.
2	1	.	7	.	.	.	11	.	9	.	.
12	4	.	2	.	.	.	.	.	6	5	.
.	1	.	.	.	.	.	.	.	28	.	.
5	3	.	3	.	.	.	.	.	8	11	.
4	.	.	.	.	1	.	.	.	3	7	15

ANN											
12	.	.	.	.	.	.	.	3	15	.	.
.	.	3	.	.	.	.	.	14	14	.	.
.	.	10	.	.	.	.	.	3	7	.	.
1	.	4	.	.	.	.	.	9	16	.	.
1	.	.	.	.	.	.	.	17	12	.	.
.	1	.	.	.	.	.	.	13	17	.	.
.	.	2	.	.	.	.	.	15	13	.	.
1	.	.	.	.	.	.	2	7	20	.	.
.	1	.	.	.	.	.	.	17	11	.	.
.	.	2	.	.	.	.	.	1	26	.	.
2	.	3	.	.	.	.	.	13	12	.	.
1	.	1	.	.	.	.	.	6	6	.	7

Table 6. AFIT Day Seven - Confusion Matrices for LBG, UFP-ONC, MLP

LBG											
28	.	.	1	.	.	.	1	.	.	.	.
1	28	.	.	.	.	.	.	1	.	.	1
.	.	17	2	.	.	1	.	.	.	.	.
1	.	.	28	.	.	.	.	1	.	.	.
.	.	2	.	28	.	.	.	.	.	.	.
.	.	.	.	.	31	.	.	.	.	.	.
.	1	.	.	.	.	22	1	.	5	1	.
.	.	.	1	.	.	.	29	.	.	.	.
1	.	.	.	.	.	.	.	28	.	.	.
.	.	.	.	.	1	.	.	.	28	.	.
4	3	2	3	.	2	.	.	.	1	15	.
1	.	.	.	.	.	.	.	.	.	.	20

UFP-ONC											
29	.	.	.	1	.	.	.	.	.	.	.
.	27	.	.	.	.	.	.	1	.	.	3
.	.	20	.	.	.	.	.	.	.	.	.
.	.	3	25	1	.	.	.	.	.	.	1
.	.	2	.	27	.	.	.	1	.	.	.
.	1	.	.	.	29	.	.	.	.	.	1
.	5	6	1	1	.	13	.	.	2	.	2
.	.	1	.	.	.	.	29	.	.	.	.
.	.	.	.	.	.	.	.	29	.	.	.
.	1	1	.	.	1	1	.	.	25	.	.
8	.	5	1	1	.	.	.	.	.	10	5
.	.	.	.	.	.	.	.	.	.	.	21

ANN											
7	.	.	.	.	1	.	.	4	18	.	.
.	7	1	.	.	.	.	.	3	18	.	2
.	.	1	.	.	.	.	.	2	15	.	2
.	.	.	1	.	.	.	.	8	21	.	.
.	2	1	.	.	1	.	.	17	7	.	2
.	.	.	.	.	21	.	.	7	3	.	.
.	.	.	.	.	1	.	.	14	12	.	3
.	.	.	.	.	.	.	13	11	6	.	.
.	.	.	.	.	.	.	.	26	3	.	.
.	.	.	.	.	.	.	.	.	29	.	.
.	1	.	.	.	1	.	.	16	11	1	.
.	.	.	.	.	.	.	.	11	7	.	3

viduals. LBG is more consistent than the UFP-ONC at this time, however the UFP-ONC has achieved very good performance with only eight cluster centers in the speaker codebooks as compared with 64 codewords in the vector quantizer codebooks. However this implementation of the UFP-ONC is not yet robust, a number of problems affect its performance. This preliminary investigation in speaker identification using fuzzy techniques indicates that further research into fuzzy based methods is warranted.

V. Conclusions

In this research the Unsupervised Fuzzy Partition-Optimal Number of Classes algorithm was implemented and applied to speaker-identification. The performance of the UFP-ONC is compared to LBG vector quantization and the multilayer perceptron.

This research demonstrated that fuzzy clustering methods can achieve the same or higher rates of identification on individuals in a database. The UFP-ONC clustering algorithm provides a very effective means of developing fuzzy speaker codebooks. The fuzzy codebooks achieve very good identification rates with fewer centroids than used in vector quantization. This comes at the expense of increased computation during training and additional storage for the fuzzy covariance matrices.

The present implementation of the UFP-ONC requires more research to achieve the robust operation of the LBG based vector quantizer. Issues that need to be addressed include :

- How to prevent repeated centroids in the Fuzzy k-means algorithm. The current implementation of the UFP-ONC does not test for coincident centroids. Coincident centroids prevent correct operation of the classifier by assigning equal memberships to each of the clusters.

- How to adapt the validity criteria to provide meaningful indicators for speech data. The cluster validity measures were not reliably generated when using speech data. When applied to simpler data sets such as the Iris data, Test6 or even RGB vectors from images, the measures were generated consistently.

- An evaluation of centroid initialization techniques. The accuracy of the FMLE algorithm depends on the accuracy of the centroids found by the FKM. The FKM also benefits from "good" initialization. This research used two techniques, the first calculated the initial centroids by adding random vectors to the mean of the data set. The second generated the initial centroids by randomly selecting ten samples from the data set and calculating the mean. Both performed these performed well, however there may be a more appropriate method which can be incorporated.

- The development of alternate fuzzy classifiers may provide more improve the identification rates to the levels attained by vector quantization.

The objective of this research was to provide and evaluation of clustering techniques to speaker identification. This objective has been met, and the results indicate that further application of fuzzy based algorithms is warranted.

Appendix A. TIMIT Sentences

The table below shows some of the phonetically balanced sentences used in the TIMIT data base.

sa1	She had your dark suit in greasy wash water all year
sa2	Don't ask me to carry an oily rag like that
sx22	When all else fails, use force
sx40	Stimulating discussions keep students' attention
si806	You need answers to four important questions
si912	You may amaze yourself and acquire a real knack for it

Appendix B. Program Listings

The following programs are written using *ANSI C* structure. The programs use no specialized interface or graphics and have executed successfully on the following platforms, SUN Sparc 2, SUN Sparc 10, IBM RS6000, Silicon Graphics Iris 4D and IBM PC (386).

The code makes extensive use of structures and functions for ease of modification and versatility. The library FuzzLIB.c provides the fuzzy k-means, the fuzzy maximum likelihood estimation and the UFP-ONC. All arrays and matrices are coded using the Numerical Recipes in C [41] format.

Also included is a MATLAB [33] script file which provides a simple implementation of the FKM and UFP-ONC code. This code was used in verification and is extremely slow for all but small data sets.

B.1 Separability C Code

```
/* ..........................................................

   Program Name : fuku.c

   Description : Calculates Separability Measure J4, see FUKUNAGA 1972

   Author :    D. Neale Prescott, (dprescot@afit.af.mil)

   University :    USAF Institute of Technology

   Date :    FEB 94

   Other Code :   a.   Numerical Recipes in C (2nd Ed)

   Note : If you don't have NRC then just replace the DSQR commands with a
          MACRO for squaring two double precision numbers.

   Input Files :   The data file in the following format. The leading comments
      are not included.


Line   1: Number of dimensions per feature vector
Line   2: Number of classes
Line   3: Number of feature vectors in class 1
Line   4: Number of feature vectors in class 2
Line   5: ...      ...        .... .. ......
Line   x: ...      ...        .... .. ......
Line   x: ...      ...        .... .. ......
Line   N: Number of feature vectors in last class
Line N+1: data for class 1 one feature vector per row
Line   x: ...      ...        .... .. ......
Line   x: ...      ...        .... .. ......
Line    : data for class 2 one feature vector per row
        .... .. ..... .. ... ...... ...... .... ..
        .... .. ..... .. ... ...... ...... .... ..
        .... .. ..... .. ... ...... ...... .... ..
        .... .. ..... .. ... ...... ...... .... ..


NOTE : The sizes of the covariance arrays is set up for
      maximum sizes of 20 dimensional feature vectors and
      a maximum of 12 classes. The "zero th" element of
      all arrays is not used.


   .......................................................... */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <errno.h>

#include "nrutil.h"

void check_input_arguments(int cmd_line, int req_args);

typedef struct statistic_params{
   double mean[21];
   double var[21];
   int samples;
   } STAT_PARAMS;

typedef struct covar_params{
   double covariance[21][21];
} COV_PARAMS;

/* ================== variables =========================== */

int dim, num_spkrs, i, j, k, p, tot;

STAT_PARAMS sp_stat[20];
STAT_PARAMS global_stat;

COV_PARAMS sp_cov[20];
```

```
COV_PARAMS S1_cov, S2_cov;

double XsubV[21], J4, J4mtx[14][14], trS1, trS2, tmp;

char fname[40];

FILE *data_in;

/* ====================== main =========================== */

main(int argc, char *argv[])
{

  /* Read in the header numbers : dimension, number of speakers,
     and features per speaker.
     ................................................... */
  check_input_arguments(argc, 2);

  sprintf(fname, "%s", argv[1]);

  data_in=fopen(fname,"r");
  if(!data_in)
  {
    printf("Cant open the file %s\n", fname);
    exit(1);
  }

  fscanf(data_in, "%d%d", &dim, &num_spkrs);
  for(i = 1; i ≤ num_spkrs; i++)
  {
    fscanf(data_in, "%d", &sp_stat[i].samples );
    for(j = 1; j ≤ dim; j++)
    {
      sp_stat[i].mean[j]  = 0.0;
      sp_stat[i].var[j]   = 0.0;
      global_stat.mean[j] = 0.0;
      global_stat.var[j]  = 0.0;
    }
  }

  /* Read in the data and calculate the means and variances
     for each of the sepakers
     ................................................... */
  tot = 0.0;
  for(i = 1; i ≤ num_spkrs; i++)
  {
    for(j = 1; j ≤ sp_stat[i].samples; j++)
    {
      for(k = 1; k ≤ dim; k++)
      {
        fscanf(data_in, "%lf", &tmp);
        sp_stat[i].mean[k]  += tmp;
        sp_stat[i].var[k]   += DSQR(tmp);
        global_stat.mean[k] += tmp;
        global_stat.var[k]  += DSQR(tmp);

      }
      tot++;
    }
  }


  /* Complete the final calculation of mean and var for each class
     ................................................... */
  for(i = 1; i ≤ num_spkrs; i++)
  {
    for(k = 1; k ≤ dim; k++)
    {
      sp_stat[i].mean[k] = sp_stat[i].mean[k]/sp_stat[i].samples;
      sp_stat[i].var[k]  = (sp_stat[i].var[k]/sp_stat[i].samples) - DSQR(sp_stat[i].mean[k]);
    }
  }
  for(j = 1; j ≤ dim; j++)
  {
    global_stat.mean[j] = global_stat.mean[j]/tot;
    global_stat.var[j]  = (global_stat.var[j]/tot) - DSQR(global_stat.mean[j]);
  }
```

```
/* calculate the covariance matrix of the class means, S1.
  It is assumed equal a priori for speakers.
  ....................................................... */
  for(i = 1; i <= dim; i++)
    for(j = 1; j <= dim; j++)
      S1_cov.covariance[i][j] = 0.0;

  for(i = 1; i <= num_spkrs; i++)
  {
    for(j = 1; j <= dim; j++)
    XsubV[j] = sp_stat[i].mean[j] - global_stat.mean[j];

    for(j = 1; j <= dim; j++)
      for(k = 1; k <= dim; k++)
      S1_cov.covariance[j][k] +=  XsubV[j]*XsubV[k];
  }


  /* Read in the data again and calculate the INTRA-CLASS covariances
    and S2.
    ............................................................... */

  rewind(data_in);
  fscanf(data_in, "%d%d", &i, &j);
  for(i = 1; i <= num_spkrs; i++)
    fscanf(data_in, "%d", &j );


  /* Read in the data and calculate the  covariance matrix
    ...................................................... */
  for(i = 1; i <= dim; i++)
    for(j = 1; j <= dim; j++)
      S2_cov.covariance[i][j] = 0.0;

  for(i = 1; i <= num_spkrs; i++)
  {
    for(j = 1; j <= dim; j++)
      for(k = 1; k <= dim; k++)
        sp_cov[i].covariance[j][k] = 0.0;

    for(j = 1; j <= sp_stat[i].samples; j++)
    {
      for(k = 1; k <= dim; k++)
      {
        fscanf(data_in, "%lf", &XsubV[k]);
        XsubV[k] = XsubV[k] - sp_stat[i].mean[k];
      }
      for(p = 1; p <= dim; p++)
        for(k = 1; k <= dim; k++)
       sp_cov[i].covariance[p][k] +=  XsubV[p]*XsubV[k];
    }
    for(j = 1; j <= dim; j++)
      for(k = 1; k <= dim; k++)
      {
        sp_cov[i].covariance[j][k] = sp_cov[i].covariance[j][k]/sp_stat[i].samples;
        S2_cov.covariance[j][k] += sp_cov[i].covariance[j][k]/num_spkrs;
      }
  }
  fclose(data_in);


  /* Calculate the trace of S1 and of S2, then calculate J4.
    For all classes.
    ...................................................... */
  trS1 = 0.0;
  trS2 = 0.0;
  J4   = 0.0;

  for(i = 1; i <= dim; i++)
  {
    trS1 += S1_cov.covariance[i][i];
    trS2 += S2_cov.covariance[i][i];
  }
  J4 = trS1/trS2;
  printf("Separability Measure for [%s] is %lf\n",fname , J4);
```

```
printf("trS1 = %4.3e"ttrS2 = %4.3e"n", trS1, trS2);


/* Calculate the trace of S1 and of S2, then calculate J4.
   For all classes. Note that only the calculations which
   affect the diagonal have been computed this time.
   ....................................................... */
for(i = 1; i ≤ num_spkrs; i++)
   for(j = 1; j ≤ num_spkrs; j++)
      J4mtx[i][j] = 0.0;

for(i = 1; i ≤ num_spkrs; i++)
{
   for(j = (i+1); j ≤ num_spkrs; j++)
   {
      trS1 = 0.0;
      trS2 = 0.0;
      for(k = 1; k ≤ dim; k++)
      {
     XsubV[k] = (sp_stat[i].mean[k] + sp_stat[j].mean[k])/2.0;
        trS1 += DSQR(sp_stat[i].mean[k] - XsubV[k]) + DSQR(sp_stat[j].mean[k] - XsubV[k]);
        trS2 += (sp_cov[i].covariance[k][k] + sp_cov[j].covariance[k][k]);
      }
      J4mtx[i][j] = trS1/trS2;
   }
}

/* Print it out ready for a Latex table
   Just copy the entire output into a Latex
   document and the table is ready to go
   .................................... */

   printf(""“begin-table"“n");
   printf(""“begin-center"“n");
   printf(""“begin-tabular"-——");
   for(i = 1; i ≤ num_spkrs; i++) printf("c");
   printf("——" “"hline “"hline “n");
   for(i = 1; i ≤ num_spkrs; i++)
     if(i ≠ num_spkrs)
       printf("$"“omega`-%d"$ & ", i);
     else
       printf("$"“omega-%d"$ ", i);
   printf(" """" “"hline “n");

   for(i = 1; i ≤ num_spkrs; i++)
   {
     for(j = 1; j ≤ num_spkrs; j++)
     {
       if(J4mtx[i][j] == 0.0)
         printf("$"“cdot$"t");
       else
         printf("%4.3f"t",J4mtx[i][j]);

       if(j == num_spkrs)
         printf("""" “n");
       else
        printf("& ");
     }
   }
   printf(""“hline “"hline “n");
   printf(""“end-tabular"“n");
   printf(""“end-center"“n");
   printf(""“label-Fukungaga J4 measure for file %s"“n", fname);
   printf(""“end-table"“n");

}

/* ====================== functions ======================= */

void check_input_arguments(int cmd_line, int req_args)
{
/* This function ensures the correct number of
   command line arguments are present.
   Neale Prescott OCT 93
*/

   if(cmd_line ≠ req_args)
```

```
    {
      printf(""nERROR Command line arguments incomplete"n");
      exit(1);
    }
  }
```

B.2 UFP-ONC C code

B.2.1 Main Program - FuzzCl.c.

```
/* ..........................................................

   Program Name : FuzzCl.c

   Description : This program provides a number of Fuzzy Clustering
      algorithms, the Fuzzy K-means (FKM) and the Fuzzy Maximum
      Likelihood Estimation (FMLE). These two are combined to form
      the Unsupervised Fuzzy Partition-Optimal Number of Classes
      (UFP-ONC) proposed by I.Gath and B.Geva IEEE Transactions
      on Pattern Analysis and Machine Intelligence, Vol 11, No.7,
      July 1989.

      The UFP-ONC will be used to determine the number and location
      of clusters in the data set. This data will then be passed to
      the Possibilistic C-Means (PCM) proposed by Krishnappuram
      and Keller, IEEE Transactions on Fuzzy Systems, Vol 1, No.2,
      May 1993, algorithm for final determination of the cluster
      centres.

      The intended purpose of this rather complicated process
      is to cluster speech data for a number of speakers involved
      in machine based speaker identification experiment for
      thesis work by Neale Prescott.

      The fuzzy clustering approach is to be compared to
      a Vector Quantisation based Speaker ID system.

      If the number of clusters is known then a faster method
      is to use the basic Fuzzy K-means and then the PCM. In this
      case the aim is to remove the heuristic way in which the
      number of clusters (or codewords) is often chosen.


   Author :    D. Neale Prescott, (dprescot@afit.af.mil)

   University :    USAF Institute of Technology

   Date :      OCT 93, 20 JAN 94

   Other Code :  a.   Numerical Recipes in C (2nd Ed)
          b.   FuzzLIB.h
          c.   FuzzLIB.c

   Input Files :   data_file, setup_file

   Output Files :    cluster_files, performance_file

   References : See Fuzzlib.c for the list.
   .......................................................... */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#include "nrutil.h"
#include "nr.h"
#include "FuzzLIB.h"

#ifdef DIAG
#define PRINT_DIAG 1
#else
#define PRINT_DIAG 0
#endif

/* ================== variables =========================== */

int status, i;

C_PARAMS config_params;
F_PARAMS feat_params;
M_PARAMS mtx_params;
S_PARAMS stats_params;
```

```
P_PARAMS perform_params;

FILE *cluster_in, *cluster_out, *perf_out;

/* ====================== main =========================== */

main(int argc, char *argv[])
{

  read_configuration_file(&config_params, &feat_params, argc, argv);

  make_matrices(&config_params, &mtx_params, &stats_params, &perform_params);

  load_features(&config_params, &mtx_params);

  get_data_stats(&config_params, &mtx_params, &stats_params);

  switch(config_params.FKM_or_UFP)
  {
    case 'U' : perform_UFP_ONC(&config_params, &mtx_params, &feat_params, &stats_params. &perform_params);
      break;

    case 'F' : initialise_clusters(&config_params, &mtx_params, &stats_params);
      calc_fuzzy_k_means(&config_params, &mtx_params, &feat_params);
            set_fuzzy_cov_identity(&mtx_params, &feat_params);
            for(i = 1;i  <= config_params.final_clusters; i++)
              print_Fuzzy_covariance(&mtx_params, &feat_params);
      save_centroids(&mtx_params, &feat_params);
      break;
  }

  if(PRINT_DIAG) print_memberships( &mtx_params, &feat_params);
  delete_matrices(&config_params, &mtx_params, &stats_params, &perform_params);

  return(0);
}
```

B.2.2 Header file - FuzzLIB.h.

```
/* ............................................
   FuzzLIB.h
   Code : ANSI C header file
   Author : D.Neale.Prescott
   Purpose : Fuzzy Clustering functions for FKM, UFP-ONC and FVQ
   Date : 20 Jan 94
   Place : AFIT, Ohio, USA
   ............................................ */
/* ================= structures ========================= */

typedef struct configuration_params{
   char comment[80];            /* A single line of comment is retained */
   char data_name[30];          /* input - data file name */
   char initial_cluster[30];  /* input - initial cluster data OPTIONAL */
   char    cluster_name[30];     /* output - cluster centres data */
   char    perform_name[30];       /* record of performance    */
   char    FKM_or_UFP;              /* select - FKM or UFP-ONC */

   int number_vectors;          /* number of input data points */
   int features_per_vector;     /* dimension of each input vector */
   int initial_clusters;      /* number of initial clusters DEFAULT 2 */
   int final_clusters;        /* Maxmimum number of clusters to be used */
   double fuzziness;
   double stop_criteria;            /* stopping point for MAX.ij(old.ij - new.ij) */
   int max_iterations;          /* If FKM does not reach stop_criteria  */
      } C_PARAMS;

typedef struct vector_params{
   int num_vcts;
   int num_ftrs;
   int num_centroids;           /* Number of centroids in use at a given time */
      } F_PARAMS;

typedef struct matrix_params{
   double **feat_mtx;             /* data samples */
   double **dist_mtx;             /* distance from each sample to each centroid */
   double **mships_mtx;             /* membership of each sample to each class */
   double **old_member_mtx;        /* previous iteration's membership */
   double **centroid_mtx;        /* locations of each centroid */
   double **F_covar_mtx;
   double aPRIORI;
   double F_det;
     }M_PARAMS;

typedef struct statistics_params{
   double *mean_vct;               /* mean vector of entire data set */
   double *var_vct;               /* variance vectorof entire data set */
     }S_PARAMS;

typedef struct performance_params{
   double *Fuzz_HV;                /* Fuzzy hypervolume - small is good */
   double *Av_Density;             /* Average cluster density - big is good */
   double *Partition_Density;         /* Partial cluster density - big is good */
   double S1;
   double S2;
     }P_PARAMS;


/* ====================================================== */

void check_input_arguments(int cmd_line, int req_args);
FILE *open_file_read(char name[]);

void int_check_range(int subject, int low, int high, char *where);
void float_check_range(float subject, float low, float high, char *where);
void zero_matrix(float **zmtx, int rows, int cols);
void zero_vector(float *zvct, int cols);
void zero_dmatrix(double **zmtx, int rows, int cols);
void zero_dvector(double *zvct, int cols);

void read_configuration_file(C_PARAMS *config_p, F_PARAMS *feat_p, int cmd_line, char *cmd_name[]);
void make_matrices(C_PARAMS *config_p, M_PARAMS *matrix_p, S_PARAMS *stats_p, P_PARAMS *perform_p);
void delete_matrices(C_PARAMS *config_p, M_PARAMS *matrix_p, S_PARAMS *stats_p, P_PARAMS *perform_p);
void load_features(C_PARAMS *config_p, M_PARAMS *matrix_p);
void load_centroids(C_PARAMS *config_p, M_PARAMS *matrix_p, double **cov_p, double *det_p, double *aprior_p);
```

```
void get_data_stats(C_PARAMS *config_p, M_PARAMS *matrix_p, S_PARAMS *stats_p);
void perform_UFP_ONC(C_PARAMS *config_p, M_PARAMS *matrix_p, F_PARAMS *feat_p, S_PARAMS *stat_p, P_PARAMS *perform_p);
void initialise_clusters(C_PARAMS *config_p, M_PARAMS *matrix_p, S_PARAMS *stat_p);

void calc_fuzzy_k_means(C_PARAMS *config_p, M_PARAMS *matrix_p, F_PARAMS *feat_p);
void copy_member_to_OLD(M_PARAMS *matrix_p, F_PARAMS *feat_p);
void calculate_dist_FKM(C_PARAMS *config_p, M_PARAMS *matrix_p, F_PARAMS *feat_p);
void compute_member_FKM(C_PARAMS *config_p, M_PARAMS *matrix_p, F_PARAMS *feat_p);
void compute_centroids_FKM(C_PARAMS *config_p, M_PARAMS *matrix_p, F_PARAMS *feat_p);
double compute_objective_FKM(M_PARAMS *matrix_p, F_PARAMS *feat_p);
double compute_AVG_objective_FKM(M_PARAMS *matrix_p, F_PARAMS *feat_p);

void calc_fuzzy_mle(C_PARAMS *config_p, M_PARAMS *matrix_p, F_PARAMS *feat_p, P_PARAMS *perform_params);
void initialise_FMLE_membs(C_PARAMS *config_p, M_PARAMS *matrix_p, F_PARAMS *feat_p);
void set_fuzzy_cov_identity(M_PARAMS *matrix_p, F_PARAMS *feat_p);
void calc_centroid_Prob(M_PARAMS *matrix_p, F_PARAMS *feat_p, int cluster);
void calc_F_covar(M_PARAMS *matrix_p, F_PARAMS *feat_p, int cluster);
void calc_dist_FMLE(M_PARAMS *matrix_p, F_PARAMS *feat_p, int cluster, P_PARAMS *perform_p);

void print_Fuzzy_covariance(M_PARAMS *matrix_p, F_PARAMS *feat_p);
void print_memberships(M_PARAMS *matrix_p, F_PARAMS *feat_p);
void save_centroids(M_PARAMS *matrix_p, F_PARAMS *feat_p);
void save_distances(C_PARAMS *config_p, M_PARAMS *matrix_p);

void calc_clust_Perform(M_PARAMS *matrix_p, P_PARAMS *perform_p, F_PARAMS *feat_p);
void store_performance_data(C_PARAMS *config_p, P_PARAMS *perform_p);
```

B.2.3 Fuzzy C Library -FuzzLIB.c.

```
/* ..........................................................

  Program Name : FuzzLIB.c

  Description : This program provides a number of Fuzzy Clustering
     algorithms, the Fuzzy K-means (FKM) and the Fuzzy Maximum
     Likelihood Estimation (FMLE). These two are combined to form
     the Unsupervised Fuzzy Partition-Optimal Number of Classes
     (UFP-ONC) proposed by I.Gath and B.Geva IEEE Transactions
     on Pattern Analysis and Machine Intelligence, Vol 11, No.7,
     July 1989.

     The intended purpose of this rather complicated process
     is to cluster speech data for a number of speakers involved
     in machine based speaker identification experiment for
     thesis work by Neale Prescott.

     The fuzzy clustering approach is to be compared to
     a Vector Quantization based Speaker ID system.


  Author :      D. Neale Prescott (dprescot@afit.af.mil)

  University :     USAF Institute of Technology

  Date :        1993-1994

  Other Code :    a.    Numerical Recipes in C (2nd Ed)
        b.    FuzzLIB.h

  References : 1.    Unsupervised optimal Fuzzy Clustering by
          I.Gath and B.Geva,  IEEE Transactions on Pattern
          Analysis and Machine Intelligence, Vol 11, No.7,
          July 1989.

           2. Pattern Recognition with Fuzzy Objective Function
          Algorithms by James Bezdek, Plenum, 1987 (2nd print)

           3. Fuzzy Clustering with a Fuzzy Covariance Matrix
          by D.E.Gustafson and W.C.Kessel
          IEEE CDC, San Diego, pp 761-766, Jan 1979

           4. Fuzzy Models For Pattern Recognition,
          Methods that Search for Structures in data.
          Editors J.C.Bezdek and S.K.Pal
          IEEE Press 1992 (Selected reprints)

  .......................................................... */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <errno.h>

#include "nrutil.h"
#include "nr.h"
#include "FuzzLIB.h"


#ifdef DIAG
#define PRINT_DIAG 1
#else
#define PRINT_DIAG 0
#endif

/* ====================== functions ======================= */

void check_input_arguments(int cmd_line, int req_args)
{
/* This function ensures the correct number of
   command line arguments are present.
   Neale Prescott OCT 93
*/
```

```
  if(cmd_line ≠ req_args)
  {
    printf("“nERROR Command line arguments incomplete“n");
    exit(1);
  }
}
/* ============================================================ */

FILE *open_file_read(char name[])
{
/* This function opens an ASCII file for reading.
  Neale Prescott OCT 93
*/
  FILE *fp;

  fp=fopen(name,"r");

  if(!fp)
  {
    printf("ERROR Cant open the file %s“n", name);
    exit(1);
  }
  return(fp);
}
/* ============================================================ */

void int_check_range(int subject, int low, int high, char *where)
{
/* This function checks the range of an integer. The USER
  specifies the LOW and HIGH values. The use of WHERE provides
  a way to write out a specific message to aid debugging.
  Neale Prescott OCT 93
*/
  if( subject < low ) printf("Number too LOW : [%d] %d : %s“n", low, subject, where);
  if( subject > high ) printf("Number too HIGH : [%d] %d : %s“n",high, subject, where);
}
/* ============================================================ */

void double_check_range(double subject, double low, double high, char *where)
{
/* This function checks the range of a double number. The USER
  specifies the LOW and HIGH values. The use of WHERE provides
  a way to write out a specific message to aid debugging.
  Neale Prescott OCT 93
*/
  if( subject < low ) printf("Number too LOW : [%lf] %lf : %s“n", low, subject, where);
  if( subject > high ) printf("Number too HIGH : [%lf] %lf : %s“n",high, subject, where);
}
/* ============================================================ */

void read_configuration_file(C_PARAMS *config_p, F_PARAMS *feat_p, int cmd_line, char *cmd_name[])
{
/* This function is specific to this Fuzzy Clustering Program. It reads in all
  the parameters required from a configuration file. This means the program can
  be run in multiple formats and reconfigured easily.
  Neale Prescott OCT 93
*/
  FILE *config_in;

  check_input_arguments(cmd_line, 2);       /* make sure the config file was on the command-line */
  config_in = open_file_read(cmd_name[1]);           /* open the configuration file */
  fgets(config_p→comment, 80, config_in);          /* read comment out of config file */
  fscanf(config_in,"%s", config_p→data_name);          /* read data in file name */
  fscanf(config_in, "%s", config_p→initial_cluster); /* read cluster in file name */
  fscanf(config_in, "%s", config_p→cluster_name);   /* read cluster out file name */
  fscanf(config_in, "%s", config_p→perform_name);     /* read perform out file name */
  fscanf(config_in, "%1s", &config_p→FKM_or_UFP );
  fscanf(config_in, "%d%d%d%d%lf%lf%d", &config_p→number_vectors,
                    &config_p→features_per_vector,
                    &config_p→initial_clusters,
                    &config_p→final_clusters,
                    &config_p→fuzziness,
                    &config_p→stop_criteria,
                    &config_p→max_iterations );

  if( (config_p→FKM_or_UFP ≠ 'F') && (config_p→FKM_or_UFP ≠ 'U') )
  {
```

```
    printf("CONFIG ERROR: Please specify Fussy k-means [F] or UFP-ONC [U]\n");
    printf("Fussy k-means is now assumed\n");
    config_p→FKM_or_UFP = 'F';
  }
  int_check_range(config_p→number_vectors,     2, 80000, "Read-CONFIG[1]");
  int_check_range(config_p→features_per_vector, 1,   200, "Read-CONFIG[2]");
  int_check_range(config_p→initial_clusters,   2,    64, "Read-CONFIG[3]");
  int_check_range(config_p→final_clusters,     2,    64, "Read-CONFIG[4]");
  int_check_range(config_p→max_iterations,   500,  3000, "Read-CONFIG[5]");

  double_check_range(config_p→fuzziness,      1.0, 3.0, "Read-CONFIG[6]");
  double_check_range(config_p→stop_criteria, 1E-6,   1, "Read-CONFIG[7]");

  feat_p→num_vcts      = config_p→number_vectors;
  feat_p→num_ftrs      = config_p→features_per_vector;
  feat_p→num_centroids = config_p→initial_clusters;
}
/* ============================================================ */
void zero_matrix(float **zmtx, int rows, int cols)
{ /* This function fills a matrix of floats with zeros. */
  int i, j;
  for(i = 1;  i ≤ rows; i++)
    for(j = 1; j ≤ cols; j++)
    zmtx[i][j] = 0.0;
}
/* ============================================================ */
void zero_vector(float *zvct, int cols)
{ /* This function fills a vector of floats with zeros. */
  int j;
  for(j = 1; j ≤ cols; j++) zvct[j] = 0.0;
}
/* ============================================================ */
void zero_dmatrix(double **zmtx, int rows, int cols)
{ /* This function fills a matrix of doubles with zeros. */
  int i, j;
  for(i = 1;  i ≤ rows; i++)
    for(j = 1; j ≤ cols; j++)
    zmtx[i][j] = 0.0;
}
/* ============================================================ */
void zero_dvector(double *zvct, int cols)
{ /* This function fills a vector of doubles with zeros. */
  int j;
  for(j = 1; j ≤ cols; j++) zvct[j] = 0.0;
}
/* ============================================================ */

void make_matrices(C_PARAMS *config_p, M_PARAMS *matrix_p, S_PARAMS *stats_p, P_PARAMS *perform_p)
{
/* This function uses Numerical Recipes in C to make
   a number of matrices for this experiment. This is
   for ease in reading the code. All matrices are initialised
   to zero. It is not a general routine.
   Neale Prescott OCT 93
*/
  matrix_p→feat_mtx       = dmatrix(1, config_p→number_vectors,     1, config_p→features_per_vector);
  matrix_p→mships_mtx     = dmatrix(1, config_p→number_vectors,     1, config_p→final_clusters);
  matrix_p→old_member_mtx = dmatrix(1, config_p→number_vectors,     1, config_p→final_clusters);
  matrix_p→centroid_mtx   = dmatrix(1, config_p→final_clusters,     1, config_p→features_per_vector);
  matrix_p→dist_mtx       = dmatrix(1, config_p→number_vectors,     1, config_p→final_clusters);
  matrix_p→F_covar_mtx    = dmatrix(1, config_p→features_per_vector, 1, config_p→features_per_vector);

  zero_dmatrix(matrix_p→feat_mtx, config_p→number_vectors, config_p→features_per_vector);
  zero_dmatrix(matrix_p→mships_mtx, config_p→number_vectors, config_p→final_clusters);
  zero_dmatrix(matrix_p→old_member_mtx, config_p→number_vectors, config_p→final_clusters);
  zero_dmatrix(matrix_p→centroid_mtx, config_p→final_clusters, config_p→features_per_vector);
  zero_dmatrix(matrix_p→dist_mtx, config_p→number_vectors, config_p→final_clusters);
  zero_dmatrix(matrix_p→F_covar_mtx, config_p→features_per_vector, config_p→features_per_vector);

  stats_p→mean_vct = dvector(1, config_p→features_per_vector);
  stats_p→var_vct  = dvector(1, config_p→features_per_vector);

  zero_dvector(stats_p→mean_vct, config_p→features_per_vector);
  zero_dvector(stats_p→var_vct, config_p→features_per_vector);

  perform_p→Fuzz_HV        = dvector(1, config_p→final_clusters);
  perform_p→Av_Density     = dvector(1, config_p→final_clusters);
```

```
  perform_p->Partition_Density = dvector(1, config_p->final_clusters);

  zero_dvector(perform_p->Fuzz_HV, config_p->final_clusters);
  zero_dvector(perform_p->Av_Density, config_p->final_clusters);
  zero_dvector(perform_p->Partition_Density, config_p->final_clusters);
}
/* ========================================================== */

void delete_matrices(C_PARAMS *config_p, M_PARAMS *matrix_p, S_PARAMS *stats_p, P_PARAMS *perform_p)
{
/* This function uses Numerical Recipes in C to free
  a number of matrices. This is for ease in reading the code.
  It is not a general routine. i.e Specific to this code.
  Neale Prescott OCT 93
*/
  free_dmatrix(matrix_p->feat_mtx,       1, config_p->number_vectors,      1, config_p->features_per_vector);
  free_dmatrix(matrix_p->mships_mtx,      1, config_p->number_vectors,      1, config_p->final_clusters);
  free_dmatrix(matrix_p->old_member_mtx, 1, config_p->number_vectors,      1, config_p->final_clusters);
  free_dmatrix(matrix_p->centroid_mtx,   1, config_p->final_clusters,     1, config_p->features_per_vector);
  free_dmatrix(matrix_p->dist_mtx,       1, config_p->number_vectors,      1, config_p->final_clusters);
  free_dmatrix(matrix_p->F_covar_mtx,    1, config_p->features_per_vector, 1, config_p->features_per_vector);

  free_dvector(stats_p->mean_vct, 1, config_p->features_per_vector);
  free_dvector(stats_p->var_vct,  1, config_p->features_per_vector);

  free_dvector(perform_p->Fuzz_HV,           1, config_p->final_clusters);
  free_dvector(perform_p->Av_Density,        1, config_p->final_clusters);
  free_dvector(perform_p->Partition_Density, 1, config_p->final_clusters);
}
/* ========================================================== */

void get_data_stats(C_PARAMS *config_p, M_PARAMS *matrix_p, S_PARAMS *stats_p)
{
/* This function calcuates the mean and variance of the data set.
  Which is in fact the initial centroid.
  The function uses Numerical Recipes in C.
  The input is a matrix of doubles (columns being features, row are feature_vectors).
  The outputs are two vectors - mean and variance.
  N.Prescott OCT 93
*/
  int col, row, num_zero;
  float average, av_dev, std_dev, var, skew, kurt, *tmp_vct;

  tmp_vct = vector(1, config_p->number_vectors);

  for(col = 1; col <= config_p->features_per_vector; col++)
  {
    num_zero = 0;
    for(row = 1; row <= config_p->number_vectors; row++)
     {
    tmp_vct[row] = (float) matrix_p->feat_mtx[row][col];
    if(tmp_vct[row] == 0.0)
      num_zero++;
     }
    if( num_zero == config_p->number_vectors)
     {
    average = 0.0;    /* This avoids an NRC error when the vecto[illegible]o */
    var = 0.0;
     }
    else
    moment( tmp_vct, config_p->number_vectors, &average, &av_dev, &std_dev, &var, &skew, &kurt);
    stats_p->mean_vct[col] = (double) average;
    stats_p->var_vct[col]  = (double) var;
  }
  free_vector( tmp_vct,1, config_p->number_vectors);
}
/* ========================================================== */

void load_features(C_PARAMS *config_p, M_PARAMS *matrix_p)
{
/* This function reads in data from an ASCII file and places it into
  "feature_matrix". The number of feature_vectors and features_per_vector
  must be specified as the data file must not have a header.
  Neale Prescott OCT 93
*/
  int row, col;
  FILE *data_in;
```

```
  data_in = open_file_read(config_p->data_name);
  for(row = 1; row <= config_p->number_vectors; row++)
    for(col = 1; col <= config_p->features_per_vector; col++)
   fscanf(data_in, "%lf", &matrix_p->feat_mtx[row][col] );

  fclose(data_in);
}
/* ========================================================== */

void load_centroids(C_PARAMS *config_p, M_PARAMS *matrix_p, double **cov_p, double *det_p, double *aprior_p)
{
/* This function reads in data from an ASCII file and places it into
   "centroid_matrix". The number of centroids and features_per_centroid
   must be specified as the data file must not have a header.
   Neale Prescott 08NOV 93
*/
  int i, row, col, cnt2;
  FILE *data_in;
  char tmp_string[80];

  data_in = open_file_read(config_p->cluster_name);
  cnt2 = 1;
  for(i = 1; i <= config_p->final_clusters; i++)
  {
    fgets(tmp_string, 80, data_in);
    fscanf(data_in, "%lf", &aprior_p[i] );
    fscanf(data_in, "%lf", &det_p[i] );
    for(row = 1; row <= config_p->features_per_vector; row++, cnt2++)
      for(col = 1; col <= config_p->features_per_vector; col++)
     fscanf(data_in, "%lf", &cov_p[cnt2][col] );
    fgets(tmp_string, 80, data_in);
  }
  fgets(tmp_string, 80, data_in);
  for(row = 1; row <= config_p->final_clusters; row++)
    for(col = 1; col <= config_p->features_per_vector; col++)
   fscanf(data_in, "%lf", &matrix_p->centroid_mtx[row][col] );

  fclose(data_in);
}
/* ========================================================== */

void perform_UFP_ONC(C_PARAMS *config_p, M_PARAMS *matrix_p, F_PARAMS *feat_p, S_PARAMS *stat_p, P_PARAMS *perform_p)
{
/* Perform the unsupervised Fuzzy Partition-Optimal number of classes
   routine. See Reference 1, listed in the header comments.
   The "fuzziness" and "stop_epsilon" are adjustable parameters
   NOTE "fuzziness > 1" and "stop_epsilon [0,1]"
   Neale Prescott OCT 93
*/
  int cluster;

  initialise_clusters(config_p, matrix_p, stat_p);

  for( cluster = config_p->initial_clusters; cluster <= config_p->final_clusters; cluster++)
  {
    if(PRINT_DIAG) printf("\n\nCentroids = %d\n\n", cluster);

    feat_p->num_centroids = cluster;

    calc_fuzzy_k_means(config_p, matrix_p, feat_p);

    if(PRINT_DIAG) save_centroids(matrix_p, feat_p);

    calc_fuzzy_mle(config_p, matrix_p, feat_p, perform_p);

    save_centroids(matrix_p, feat_p);
  }
  if(PRINT_DIAG)  store_performance_data(config_p, perform_p);
}
/* ========================================================== */

void initialise_clusters(C_PARAMS *config_p, M_PARAMS *matrix_p, S_PARAMS *stats_p)
{
/* Refer to pp775, Section 2.B of the UFP-ONC paper for why
   the 2nd centroid is chosen this way.
   Initial cluster centre is at the data mean.
```

```
  Consecutive centres are (+/-) Ramdom amounts of Standard Deviation from mean
  Neale Prescott 20JAN94.
*/
  int cluster, feature, pseudo, i;
  double mult;

  srand(1);
  for(cluster = 1; cluster <= config_p->final_clusters; cluster++)
  {
    if( cluster == 1)
      for(feature = 1; feature <= config_p->features_per_vector, feature++)
        matrix_p->centroid_mtx[cluster][feature] = stats_p->mean_vct[feature];
    else
    for(i=1; i <= 10; i++)
    {
      pseudo = (rand() % config_p->number_vectors) + 1;
      for(feature = 1; feature <= config_p->features_per_vector; feature++)
        matrix_p->centroid_mtx[cluster][feature] += matrix_p->feat_mtx[pseudo][feature]/10.0;
    }


  }
}
/* ======================================================== */

void calc_fuzzy_k_means(C_PARAMS *config_p, M_PARAMS *matrix_p, F_PARAMS *feat_p)
{
/* Refer to pp774, Section 2.A of the UFP-ONC paper eqns 1,2,3,4,5
  See Reference 2
  Neale Prescott OCT 93.
*/
  int iterations = 0;
  double objective_fn;

  calculate_dist_FKM(config_p, matrix_p, feat_p);
  compute_member_FKM(config_p, matrix_p, feat_p);
  do
  {
    compute_centroids_FKM(config_p, matrix_p, feat_p);
    copy_member_to_OLD(matrix_p, feat_p);
    calculate_dist_FKM(config_p, matrix_p, feat_p);
    compute_member_FKM(config_p, matrix_p, feat_p);
    objective_fn = compute_objective_FKM(matrix_p, feat_p);
    iterations++;
    if(PRINT_DIAG) printf("FKM Iterations : %4d : Obj = %lf"n", iterations, objective_fn);
    if( iterations == config_p->max_iterations)
   printf("FKM - maximum iterations [%d] reached. Stopping."n", iterations );
  }
  while( (objective_fn > config_p->stop_criteria) && (iterations < config_p->max_iterations) );
}
/* ======================================================== */

void copy_member_to_OLD(M_PARAMS *matrix_p, F_PARAMS *feat_p)
{
/* 20 OCT 93 : The tricky use of pointers has replaced the element by
            element copying.
  NOTE : That the memberships matrix will contain old data,
       care must be taken not to use these.
  Neale Prescott OCT 93
*/
  double **tmp_ptr;

  tmp_ptr = matrix_p->old_member_mtx;
  matrix_p->old_member_mtx = matrix_p->mships_mtx;
  matrix_p->mships_mtx = tmp_ptr;
}
/* ======================================================== */

void calculate_dist_FKM(C_PARAMS *config_p, M_PARAMS *matrix_p, F_PARAMS *feat_p)
{
/* This function computes the distance from every sample to each of the
  centroids. The values are stored in the matrix "dist_mtx".
  SQR(a) is defined in nrutil, Numerical Recipes
  Neale Prescott OCT 93
*/
  double *tmp_dst, dist;
  int row, cl, ft;
```

```
  tmp_dst = dvector(1, config_p->features_per_vector);
  for(row = 1; row <= feat_p->num_vcts; row++)
    for(cl = 1; cl <= feat_p->num_centroids; cl++)
    {
      for(ft = 1; ft <= feat_p->num_ftrs; ft++)
     tmp_dst[ft] = matrix_p->feat_mtx[row][ft] - matrix_p->centroid_mtx[cl][ft];

    dist = 0.0;
    for(ft = 1; ft <= feat_p->num_ftrs; ft++)
      dist += DSQR(tmp_dst[ft]);

    matrix_p->dist_mtx[row][cl] = dist;
    }
  free_dvector(tmp_dst, 1, feat_p->num_ftrs);
}
/* ====================================================== */

void compute_member_FKM(C_PARAMS *config_p, M_PARAMS *matrix_p, F_PARAMS *feat_p)
{
/* This function computes the membership of every data sample to each
  of the clusters (classes). See Eqn.2 of Reference 1.

  Neale Prescott OCT 93, FEB 94
*/
  int row, cl, FLAG;
  double sum_dst;

  for(row = 1; row <= feat_p->num_vcts; row++)
  {
    sum_dst = 0.0;
    FLAG = -1;
    for(cl = 1; cl <= feat_p->num_centroids; cl++)
    {
      if(matrix_p->dist_mtx[row][cl] != 0.0)
     sum_dst += (1.0/matrix_p->dist_mtx[row][cl]);
      else
        FLAG = cl;
    }
    for(cl = 1; cl <= feat_p->num_centroids; cl++)
    {
      if(FLAG == -1)
        matrix_p->mships_mtx[row][cl] =  (1.0/matrix_p->dist_mtx[row][cl])/sum_dst;
      else
        if(cl == FLAG)
          matrix_p->mships_mtx[row][cl] = 1.0;   /* This point is at the centroid. Membership must be 1 */
        else
          matrix_p->mships_mtx[row][cl] = 0.0;   /* Must have zero membership in all other clusters */
    }
  }
}
/* ====================================================== */

void compute_centroids_FKM(C_PARAMS *config_p, M_PARAMS *matrix_p, F_PARAMS *feat_p)
{
/* Computes the new centroids of the Fuzzy K Means as per eqn.3 of
  Reference.1
  DANGER : If the denominator is zero this will explode. But it should
     be "impossible" for this to occur!
  Note : "eta" is a cluster validity check from a paper by Krisnapuram and Keller
        IEEE Trans on Fuzzy Systems Vol.1, No.2, May 1993.

  Neale Prescott OCT 93
*/

  int row, cl, ft;
  double *numerator;
  double tmp_memb,  denominator, eta;

  numerator = dvector(1, feat_p->num_ftrs);
  for(cl = 1; cl <= feat_p->num_centroids; cl++)
  {
    zero_dvector(numerator, feat_p->num_ftrs);
    denominator = 0.0;
    eta = 0.0;
    for(row = 1; row <= feat_p->num_vcts; row++)
    {
```

```
    tmp_memb = DSQR(matrix_p→mships_mtx[row][cl]);
      eta += (tmp_memb * matrix_p→dist_mtx[row][cl]);
    for(ft = 1; ft ≤ feat_p→num_ftrs; ft++)
      numerator[ft] += tmp_memb * matrix_p→feat_mtx[row][ft];
    denominator += tmp_memb;
     }
     for(ft = 1; ft ≤ feat_p→num_ftrs; ft++)
    matrix_p→centroid_mtx[cl][ft] = numerator[ft]/denominator;
     eta = eta / denominator;
  }
  free_dvector(numerator, 1, feat_p→num_ftrs);
}
/* ========================================================== */

double compute_objective_FKM(M_PARAMS *matrix_p, F_PARAMS *feat_p)
{
/* See eqn.4 of Reference 1.
  DMAX(a,b) is in nrutil, Numerical Recipes)
  Neale Prescott OCT 93
*/
  double maximum = 0.0, difference;
  int row, cl;

  for(row = 1; row ≤ feat_p→num_vcts; row++)
    for(cl = 1; cl ≤ feat_p→num_centroids; cl++)
     {
    difference = fabs( matrix_p→mships_mtx[row][cl] - matrix_p→old_member_mtx[row][cl]);
    maximum = DMAX(maximum, difference);
     }
  return(maximum);
}
/* ========================================================== */

void print_memberships( M_PARAMS *matrix_p, F_PARAMS *feat_p)
{
/* This function prints out the memberships for each data point
  and is primarily for de-bugging. However it could be quickly
  modified for use in Gnuplot.
  Neale Prescott OCT 93
*/
  int i, j;

  printf("# Memberships for [%d] centroids "n", feat_p→num_centroids);
    for(i = 1; i ≤ feat_p→num_vcts; i++)
  {
    for( j = 1; j ≤ feat_p→num_centroids; j++)
      printf("%lf  ", matrix_p→mships_mtx[i][j] );
    printf(""n");
  }
}
/* ========================================================== */

void save_centroids(M_PARAMS *matrix_p, F_PARAMS *feat_p)
{
/* The final cluster coordinates must be saved. This is because the UFP-ONC
  algorithm starts with two centroids then increases up to the MAX_CLUST.
  Once the optimum number of classes is found we would like to locate
  the cluster coordinates without re-calculating them.

  NOTE : Modify this to send it to the file specified in the configuration file
  Neale Prescott OCT 93
*/

  int cl, ft;

  printf("# Cluster locations for [%d] centroids of [%d] dimensions : "n", feat_p→num_centroids, feat_p→num_ftrs);
  for(cl = 1; cl ≤ feat_p→num_centroids; cl++)
  {
    for( ft = 1; ft ≤ feat_p→num_ftrs; ft++)
      printf("%e  ", matrix_p→centroid_mtx[cl][ft] );
    printf(""n");
  }
}
/* ========================================================== */

void save_distances(C_PARAMS *config_p, M_PARAMS *matrix_p)
{
```

```
/* For debugging purposes
   Neale Prescott OCT 93
*/
   int row, cl;

   printf("Distances from  centoids"n");
   for(row = 1; row ≤ config_p→number_vectors; row++)
   {
      for(cl = 1; cl ≤ config_p→final_clusters; cl++)
         printf("%lf  ", matrix_p→dist_mtx[row][cl]);
      printf(""n");
   }
}
/* ========================================================== */

void calc_fuzzy_mle(C_PARAMS *config_p, M_PARAMS *matrix_p, F_PARAMS *feat_p, P_PARAMS *perform_p)
{
/* This function performs the Fuzzy Maximum Likelihood Estimate
   as per Reference 1.
   Note : Due to the size and number of Fuzzy Covariance matrices
   this function works on one cluster (centroid) at a time to
   limit the amount of memory required.
   Neale Prescott OCT 93
*/
   double  objective_fn;
   int  iterations = 0,
    cluster,i,j;

   initialise_FMLE_membs(config_p, matrix_p, feat_p);
   set_fuzzy_cov_identity(matrix_p, feat_p);
   for(cluster = 1; cluster ≤ feat_p→num_centroids; cluster++)
   {
      calc_centroid_Prob(matrix_p, feat_p, cluster);
      calc_dist_FMLE(matrix_p, feat_p, cluster, perform_p);
   }
   compute_member_FKM(config_p, matrix_p, feat_p);

   do
   {
      compute_centroids_FKM(config_p, matrix_p, feat_p);
      for(cluster =1; cluster ≤ feat_p→num_centroids; cluster++)
      {
         calc_F_covar(matrix_p, feat_p, cluster);
     calc_centroid_Prob(matrix_p, feat_p, cluster);
     calc_dist_FMLE(matrix_p, feat_p, cluster, perform_p);
      }
      copy_member_to_OLD(matrix_p, feat_p);
      compute_member_FKM(config_p, matrix_p, feat_p);
      objective_fn = compute_objective_FKM(matrix_p, feat_p);

      iterations++;
      if( iterations == config_p→max_iterations)
     printf("FMLE - maximum iterations [%d] reached. Stopping."n", iterations );
      if(PRINT_DIAG) printf("FMLE : [%4d]  Obj: %lf"n", iterations, objective_fn);
   }
   while( (objective_fn > config_p→stop_criteria ) && (iterations < config_p→max_iterations) );

   calc_clust_Perform(matrix_p, perform_p, feat_p);
}
/* ========================================================== */

void initialise_FMLE_membs(C_PARAMS *config_p, M_PARAMS *matrix_p, F_PARAMS *feat_p)
{
/* This function initialises the memberships to (1/num Of cluster) for
   the first iteration of the FMLE. See Gustafson-Kessel, or
   "Fitting an Unknown Number of Lines and Planes to Image Data through
    Compatible Cluster Merging" by R. Krishnapuram and C-H Freg, Pattern Recognition,
    Vol.25, pp. 385-400, 1992.

   Neale Prescott Feb 94
*/

   int i, j;
   double MEMB;

   MEMB = 1.0/((double)feat_p→num_centroids);
   for(i = 1; i ≤ config_p→number_vectors; i++)
```

```
    for(j = 1; j ≤ feat_p→num_centroids; j++)
      matrix_p→mships_mtx[i][j] = MEMB;
}
/* ========================================================================= */

void set_fuzzy_cov_identity(M_PARAMS *matrix_p, F_PARAMS *feat_p)
{
/* This function initialises the Fuzzy Covariance Matrix to the Identity matrix for
   the first iteration of the FMLE. See Gustafson-Kessel, or
   "Fitting an Unknown Number of Lines and Planes to Image Data through
    Compatible Cluster Merging" by R. Krishnapuram and C-H Freg, Pattern Recognition,
    Vol.25, pp. 385-400, 1992.

   Neale Prescott Feb 94
*/

   int i, j;

   for(i = 1; i ≤ feat_p→num_ftrs; i++)
     for(j = 1; j ≤ feat_p→num_ftrs; j++)
       if( i ≠ j)
         matrix_p→F_covar_mtx[i][j] = 0.0;
       else
         matrix_p→F_covar_mtx[i][j] = 1.0;
   matrix_p→F_det = 1.0;
}
/* ========================================================================= */

void calc_centroid_Prob(M_PARAMS *matrix_p, F_PARAMS *feat_p, int cluster)
{
/* This function calculates the a priori probability of selecting the "i"th clsuter
   Refer tp eqn(8), pp774, of Reference 1.
   Neale Prescott OCT 93
*/
   int i;
   double P_i = 0.0;

   for(i = 1; i ≤ feat_p→num_vcts; i++)
      P_i += matrix_p→mships_mtx[i][cluster];
   matrix_p→aPRIORI = P_i / ((double)feat_p→num_vcts);
}
/* ========================================================================= */

void calc_F_covar(M_PARAMS *matrix_p, F_PARAMS *feat_p, int cluster)
{
/* This function calculates the Fuzzy Covariance Matrix for the specified cluster.
   This is shown at pp.775, eqn(9) of Reference 1.
   The membership matrix is now used to hold the posteriori probabiltities, h(i|Xj) (see pp774)
   Numerical Recipes in C are used to find the inverse and determinant F^(-1) and |F|
   Neale Prescott OCT 93
*/

   double *XsubV, **tmp_F, denom, *w, **v, wmax, wmin, tmp_det, d;
   int row, ft, i, j, k;

   XsubV = dvector(1, feat_p→num_ftrs);                    /* vector from sample to current centroid */
   tmp_F = dmatrix(1, feat_p→num_ftrs, 1, feat_p→num_ftrs );
   w = dvector(1, feat_p→num_ftrs);                        /* SVD requirement */
   v = dmatrix(1, feat_p→num_ftrs, 1, feat_p→num_ftrs );   /* SVD requirement */

   zero_dvector(XsubV, feat_p→num_ftrs);
   zero_dmatrix(tmp_F, feat_p→num_ftrs, feat_p→num_ftrs);

/* calc the Fuzzy covariance matrix for current cluster */
/*.....................................................*/
   denom = 0.0;
   for(row = 1; row ≤ feat_p→num_vcts; row++)
   {
     zero_dvector(XsubV, feat_p→num_ftrs);
     for(ft = 1; ft ≤ feat_p→num_ftrs; ft++)
     XsubV[ft] = matrix_p→feat_mtx[row][ft] - matrix_p→centroid_mtx[cluster][ft];
      for(i = 1; i ≤ feat_p→num_ftrs; i++)
     for(j = 1; j ≤ feat_p→num_ftrs; j++)
       tmp_F[i][j] += (XsubV[i]*XsubV[j] * matrix_p→mships_mtx[row][cluster]);

      denom += matrix_p→mships_mtx[row][cluster];
   }
```

```
   for(i = 1; i ≤ feat_p→num_ftrs; i++)
      for(j = 1; j ≤ feat_p→num_ftrs; j++)
     tmp_F[i][j] = tmp_F[i][j]/denom;


/* Determinant and Inverse Calculation using Singular Value Decomposition */
/* ........................................................................ */

/* Decompose Fuzzy covariance matrix into (2) orthonormal and a diagonal matrix */
  dsvdcmp( tmp_F, feat_p→num_ftrs, feat_p→num_ftrs, w, v );

/* Transpose the U matrix which was returned in tmp_F */
   for(i = 1; i ≤ feat_p→num_ftrs; i++)
      for(j = i; j ≤ feat_p→num_ftrs; j++)
         if( i ≠ j)
         {
            d = tmp_F[i][j];
            tmp_F[i][j] = tmp_F[j][i];
            tmp_F[j][i] = d;
         }

/* Multiply the inverse diagonal (1/w) by transpose U */
   for(i = 1; i ≤ feat_p→num_ftrs; i++)
      for(j = 1; j ≤ feat_p→num_ftrs; j++)
         tmp_F[i][j] = tmp_F[i][j] / w[i];

/* Perform final matrix calculation to find Inverse of Fuzzy Covariance */
   zero_dmatrix(matrix_p→F_covar_mtx, feat_p→num_ftrs, feat_p→num_ftrs);
   for(i = 1; i ≤ feat_p→num_ftrs; i++)
      for(j = 1; j ≤ feat_p→num_ftrs; j++)
         for(k = 1; k ≤ feat_p→num_ftrs; k++)
            matrix_p→F_covar_mtx[i][j] += (v[i][k] * tmp_F[k][j]);

/* Determine the CONDITION number of the SVD */
   wmax = 0.0;
   for(j=1; j ≤ feat_p→num_ftrs; j++)
      if(w[j] > wmax)
         wmax = w[j];

   wmin = 1.0e20;
   for(j = 1; j ≤ feat_p→num_ftrs; j++)
      if(w[j] < wmin) wmin = w[j];

/* Refer to Numerical Recipes in C. Basically indicates near singular matrix */
/*   for(j = 1; j ≤ feat_p->num_ftrs; j++)
      if(w[j] < (wmax * 1.0e-6)) w[j] = 0.0;
*/
   if(PRINT_DIAG) printf("Condition No. : %lf\n", wmax/wmin );

/* Calculate determinant of Fuzzy Covariance matrix              */
/* Note : detA = det(invA)                                       */
/* The diagonal matrix contains the eigenvalues                  */
/* The product of the eigenvalues is the determinant             */
/* See Linear Algebra by Strang 1988   and NRC 2nd Ed.           */
/* Logs have been used to reduce risk due to over/underflow      */

   matrix_p→F_det = 0.0;
   for(j = 1; j ≤ feat_p→num_ftrs; j++)
      if(w[j] ≠ 0.0)
         matrix_p→F_det += log10( w[j] );
   matrix_p→F_det = pow(10.0, matrix_p→F_det);

/* Delete all temporary matrices and vectors */
   free_dmatrix(tmp_F, 1, feat_p→num_ftrs, 1, feat_p→num_ftrs );
   free_dvector(w, 1, feat_p→num_ftrs);
   free_dmatrix(v, 1, feat_p→num_ftrs, 1, feat_p→num_ftrs );
   free_dvector(XsubV, 1, feat_p→num_ftrs);
}
/* ============================================================ */

void print_Fuzzy_covariance(M_PARAMS *matrix_p, F_PARAMS *feat_p)
{
/* Two puposes : 1. Save Fuzzy COvariance Matrix for use in the Classifier
               2. Debugging
   Neale Prescott  OCT 93
*/
```

```
  int i, j;
  printf("# Inverted Fuzzy Covariance Matrix [%d X %d]\n", feat_p->num_ftrs, feat_p->num_ftrs);
  printf("%e\n", matrix_p->aPRIORI);
  printf("%e\n", matrix_p->F_det);
  for(i = 1;i <= feat_p->num_ftrs; i++)
  {
    for(j=1;j<=feat_p->num_ftrs;j++)
   printf("%e ", matrix_p->F_covar_mtx[i][j]);
    printf("\n");
  }
}
/* ============================================================== */

void calc_dist_FMLE(M_PARAMS *matrix_p, F_PARAMS *feat_p, int cluster, P_PARAMS *perform_p)
{
/* This function calculates the distance for each sample to the current cluster
  See Reference 1, pp774, eqn(7).
  It also passes partial results to the performance calc, refer eqns(12)(14) on pp775.
  Neale Prescott OCT 93
*/

  double *XsubV, *tmpR, mult;
  double dist;
  int row, i, j;

  XsubV = dvector(1, feat_p->num_ftrs);          /* vector from sample to current centroid */
  tmpR  = dvector(1, feat_p->num_ftrs);
  zero_dvector(XsubV, feat_p->num_ftrs);
  zero_dvector(tmpR,  feat_p->num_ftrs);

  for(row = 1; row <= feat_p->num_vcts; row++)
  {
    for(i = 1; i <= feat_p->num_ftrs; i++)
   XsubV[i] = matrix_p->feat_mtx[row][i] - matrix_p->centroid_mtx[cluster][i];

    for(i = 1; i <= feat_p->num_ftrs; i++)
    {
   tmpR[i] = 0.0;
   for(j = 1; j <= feat_p->num_ftrs; j++)
     tmpR[i] += (matrix_p->F_covar_mtx[i][j]*XsubV[j]);
    }
    dist = 0.0;
    for(i = 1; i <= feat_p->num_ftrs; i++)
   dist = dist + (XsubV[i]*tmpR[i]);
    if(dist < 0.0)
    {
      printf("OVERFLOW : NEGATIVE FMLE DIST %lf\n\n",dist);
      dist = 0.0;
    }

    if(dist < 1.0)
    {
   perform_p->S1 += matrix_p->mships_mtx[row][cluster];
      if(PRINT_DIAG) printf("Distance is inside ellipsoid %e\n",dist);
    }

    mult = (log10(matrix_p->F_det))/2.0; /* Take "sqrt" hopefully without underflow */

    mult = (pow(10.0, mult))/matrix_p->aPRIORI;

    dist = mult * exp(dist/2.0);

    matrix_p->dist_mtx[row][cluster] =  dist;
  }
  free_dvector(XsubV, 1, feat_p->num_ftrs);
  free_dvector(tmpR,  1, feat_p->num_ftrs);
}
/* ============================================================== */

void calc_clust_Perform(M_PARAMS *matrix_p, P_PARAMS *perform_p, F_PARAMS *feat_p )
{
/*  See Krishnapuram and Freg, Pattern Recognition, vol25 1992 or
   Bezdek, or Bain. And Gath and Geva in Reference 1.

  Neale Prescott Feb 94
*/
```

```
  int i, j, Cl_cnt;

  perform_p→Fuzz_HV[feat_p→num_centroids] = 0.0;
  perform_p→S2 = 0.0;
  perform_p→Av_Density[feat_p→num_centroids] = 0.0;

  for(Cl_cnt = 1; Cl_cnt ≤ feat_p→num_centroids; Cl_cnt++)
  {
    calc_F_covar(matrix_p, feat_p, Cl_cnt);
    print_Fuzzy_covariance(matrix_p, feat_p);
    perform_p→Fuzz_HV[feat_p→num_centroids] += sqrt(matrix_p→F_det);

    perform_p→S1 = 0.0;
    calc_centroid_Prob(matrix_p, feat_p, Cl_cnt );
    calc_dist_FMLE(matrix_p, feat_p, Cl_cnt, perform_p);
    perform_p→S2 += perform_p→S1;
    perform_p→Av_Density[feat_p→num_centroids] += (perform_p→S1/sqrt(matrix_p→F_det));
  }
  perform_p→Av_Density[feat_p→num_centroids] = perform_p→Av_Density[feat_p→num_centroids]/feat_p→num_centroids;
  perform_p→Partition_Density[feat_p→num_centroids] = (perform_p→S2/perform_p→Fuzz_HV[feat_p→num_centroids]);
  if(PRINT_DIAG) printf("Hypervolume = %e   S2 = %e"n", perform_p→Fuzz_HV[feat_p→num_centroids], perform_p→S2 );
}
/* ========================================================== */

void store_performance_data(C_PARAMS *config_p, P_PARAMS *perform_p)
{
/* This function output the Performance results for all the
   numbers of clusters evaluated
   Neale Prescott OCT 93
*/
  int clust;

  printf(""n# Cluster"tFuzzy Hypervol"tAv Density"tPartition Density"n");
  printf( "# ......."t............."t........."t................"n");
  for(clust = config_p→initial_clusters; clust ≤ config_p→final_clusters; clust++)
   printf("   %d"t"t%e"t%e"t%e"n", clust, perform_p→Fuzz_HV[clust], perform_p→Av_Density[clust], perform_p→Partition_Density[clust]
);
}
```

B.2.4 Fuzzy Classifier - FVQ.c.

```
/* ........................................................

   Program Name : FVQ.c

   Description : This is a modification of FuzzCl.c to perform
                 as a Fuzzy Vector Quantiser. 08 NOV 93.

   Author :      Neale Prescott

   University :     USAF Institute of Technology

   Date :        08 NOV 93

   Other Code :  Numerical Recipes in C (2nd Ed)
                 FuzzLIB.c
                 FuzzLIB.h

   Input Files :  data_file, setup_file

   References : 1.   Unsupervised optimal Fuzzy Clustering by
          I.Gath and B.Geva,  IEEE Transactions on Pattern
          Analysis and Machine Intelligence, Vol 11, No.7,
          July 1989.

           2.   A Possibilistic Approach to Clustering  by
          Krishnappuram and Keller, IEEE Transactions on Fuzzy
          Systems, Vol 1, No.2, May 1993.

                3. Speech Coding Method Using Fuzzy Vector Quantization
                   by  ASAKAWA, ICHIKAWA, YAJIMA and YAMASAKI
                   IEEE ICASSP 1989, pp755 -pp758

                4. A 2.4 KBPS Speech Coding Method Based on Fuzzy Vector
                   Quantization by ASAKAWA, YAMASAKI and ICHIKAWA
                   IEEE ICASSP 1990, pp673-676

  ........................................................ */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <errno.h>
#include "FuzzLIB.h"
#include "nrutil.h"

#ifdef DIAG
#define PRINT_DIAG 1
#else
#define PRINT_DIAG 0
#endif

/* ============================================================ */
void FMLE_vector_quantise(C_PARAMS *config_p, M_PARAMS *matrix_p, F_PARAMS *feat_p,
                    P_PARAMS *perform_p, double **cov_p, double *det_p, double *aprior_p);

/* ================== variables ============================ */

int status;

C_PARAMS config_params;
F_PARAMS feat_params;
M_PARAMS mtx_params;
S_PARAMS stats_params;
P_PARAMS perform_params;

double **COV_mtx, *DET_mtx, *PRIORI_mtx;

FILE *cluster_in;

/* ====================== main ============================ */

main(int argc, char *argv[])
{
```

```
  read_configuration_file(&config_params, &feat_params, argc, argv);

  make_matrices(&config_params, &mtx_params, &stats_params, &perform_params);

  COV_mtx = dmatrix(1, (config_params.features_per_vector*config_params.final_clusters) ,
              1, config_params.features_per_vector );

  PRIORI_mtx = dvector(1, config_params.final_clusters );

  DET_mtx = dvector(1, config_params.final_clusters );

  load_features(&config_params, &mtx_params);

  load_centroids(&config_params, &mtx_params, COV_mtx, DET_mtx, PRIORI_mtx);

  FMLE_vector_quantise(&config_params, &mtx_params, &feat_params, &perform_params,
                  COV_mtx, DET_mtx, PRIORI_mtx);

  delete_matrices(&config_params, &mtx_params, &stats_params, &perform_params);

  free_dmatrix(COV_mtx, 1, (config_params.features_per_vector*config_params.final_clusters) ,
                  1, config_params.features_per_vector );

  free_dvector(PRIORI_mtx, 1, config_params.final_clusters );

  free_dvector(DET_mtx, 1, config_params.final_clusters );

  return(0);
}
/* ======================================================= */


void FMLE_vector_quantise(C_PARAMS *config_p, M_PARAMS *matrix_p, F_PARAMS *feat_p,
                  P_PARAMS *perform_p, double **cov_p, double *det_p, double *aprior_p)
{
  int cluster, i, j, k, best;
  double  tmp_dist, dist, mean_dist, var_dist, tot_memb, max_memb, mean_memb;


  k = 1;
  for(cluster =1; cluster ≤ feat_p→num_centroids; cluster++)
  {
    for(i = 1; i ≤ config_p→features_per_vector; i++, k++)
      for(j = 1; j ≤ config_p→features_per_vector; j++)
     matrix_p→F_covar_mtx[i][j] = cov_p[k][j];

    matrix_p→F_det = det_p[cluster];
    matrix_p→aPRIORI = aprior_p[cluster];

    calc_dist_FMLE(matrix_p, feat_p, cluster, perform_p);
  }
  compute_member_FKM(config_p, matrix_p, feat_p);

  dist = 0.0;
  var_dist = 0.0;
  tot_memb = 0.0;
  for(i = 1; i ≤ config_p→number_vectors; i++)
  {
    max_memb = 0.0;
    for(j = 1; j ≤ config_p→final_clusters; j++)
   if(max_memb < matrix_p→mships_mtx[i][j])
      {
        max_memb = matrix_p→mships_mtx[i][j];
        best = j;
      }
     tot_memb += max_memb;
     tmp_dist = (log(matrix_p→dist_mtx[i][best]));
     var_dist += DSQR(tmp_dist);
     dist += tmp_dist;
  }

  mean_memb = tot_memb / config_p→number_vectors ;
  mean_dist = dist / config_p→number_vectors;
  var_dist = (var_dist/config_p→number_vectors) – DSQR(mean_dist);
  var_dist = sqrt(var_dist);
```

```
printf("Distortion [%s] for [%s] \t: %e\t%e\t%e\t%e\n", config_p->data_name,
                                        config_p->cluster_name,
                                        dist,
                                        mean_dist,
                                        (mean_dist + 3*var_dist),
                                        mean_memb);

}
/* ============================================================ */
```

B.2.5 makefile.

```
# FLTLT D.N.Prescott
# Thesis Fuzzy Clustering
# 08 OCT 93
# FILE: makefile for Fuzzy Code for THESIS
# PURPOSE: This file will allow automating building of the
#          executable programs defined.

# The best reference for MAKE is "Managing Projects with make" by Oram and Talbot
# O'Reilly & Associates, Inc.


DEBUG_FLAG   = -g  # allows debugging to work
MATH_LIB     = -lm # location of C math libraries
GCC          = gcc     # location of ANSI-C compiler on Hawkeye
DNP_FLAG = -DDIAG # -DDIAG Private debugging flag for verbose mode
#
#.......................................................................

F2 :F2.c nr.h FuzzLIB.o nrutil.o moment.o dpythag.o dsvdcmp.o
${GCC} -o F2  F2.c FuzzLIB.o nrutil.o moment.o dpythag.o dsvdcmp.o ${DEBUG_FLAG} ${MATH_LIB} ${DNP_FLAG}

FVQ :FVQ.c FuzzLIB.o nrutil.o moment.o dpythag.o dsvdcmp.o
gcc -o FVQ FVQ.c FuzzLIB.o nrutil.o moment.o dpythag.o dsvdcmp.o -lm $(cflags) -DSUN -DDIAG


fuku : fuku.c FuzzLIB.o  nrutil.o
${GCC} -o fuku fuku.c nrutil.o ${MATH_LIB} ${DNP_FLAG} ${DEBUG_FLAG}



# maximally optimised version of F2 for SUN
# Note whole compilation is without debug. ie Fuzzlib.c is optimised too.
#-----------------------------------------------------------------------
F2opt :F2.c nr.h FuzzLIB.c nrutil.o moment.o dpythag.o dsvdcmp.o
${GCC} -o F2opt F2.c FuzzLIB.c nrutil.o moment.o dpythag.o dsvdcmp.o ${MATH_LIB} -O2

FVQopt :FVQ.c FuzzLIB.o nrutil.o moment.o dpythag.o dsvdcmp.o
gcc -O2 -o FVQopt FVQ.c FuzzLIB.c nrutil.o moment.o dpythag.o dsvdcmp.o -lm -DSUN


# Library routines
#-----------------------------------------------------------------------

FuzzLIB.o :FuzzLIB.c FuzzLIB.h nrutil.h nr.h
${GCC}  -c FuzzLIB.c ${DEBUG_FLAG} ${DNP_FLAG}


nrutil.o : nrutil.c
${GCC} -c nrutil.c ${DEBUG_FLAG}

moment.o : moment.c nrutil.h
${GCC} -c moment.c ${DEBUG_FLAG}

dpythag.o : dpythag.c nrutil.h
${GCC} -c dpythag.c ${DEBUG_FLAG}

dsvdcmp.o : dsvdcmp.c nrutil.h
${GCC} -c dsvdcmp.c ${DEBUG_FLAG}
```

B.3 Simple UFPONC in MATLAB script

```
% A quick sanity check of FKM and UFPONC
% Neale Prescot
%----------------------------------------

% Global constants
%-----------------
max_centroids = 4;
epsilon = 0.001;
max_iter = 500;


load  test6.dat
data =test6;
clear test6
[m,n] =size(data);
%format short e

% Make the necessary matrices
%----------------------------
centroids = zeros(max_centroids, n);
distances = zeros(m, max_centroids);
membships = zeros(m, max_centroids);
oldships  = zeros(m, max_centroids);

% Initialise the centroids
%-------------------------
first_cen = mean(data);
centroids(1,:)=first_cen;
centroids(2,:) = first_cen + (first_cen * (-1)^1 * 0.005);
centroids(3,:) = first_cen + (first_cen * (-1)^2 * 0.005);
clear first_cen

%centroids(1,:) = data(5,:);
%centroids(2,:) = data(17,:);

% Initialise some variables
%--------------------------
objective = 10;
CLUST = 3;
track = [];
iterations = 0;


   % Calculate the distances
   %------------------------
      for i=1:m
      for j=1:CLUST
         XsubV = data(i,:) - centroids(j,:);
         tmp = XsubV * XsubV';
         distances(i,j) =  tmp;
      end
   end

   % Calculate the  memberships
   %----------------------------
   for i=1:m
      sum_all = 0.0;
      FLAG = -1;
      for j=1:CLUST
         if( distances(i,j) ~= 0.0)
```

```
          sum_all = sum_all + 1.0/distances(i,j);
        else
          FLAG = j;
        end
      end

      for j=1:CLUST
        if(FLAG == -1)
          membships(i,j) = (1.0/distances(i,j))/sum_all;
        else
          if(j == FLAG)
            membships(i,j) = 1.0;
          else
            membships(i,j) = 0.0;
          end
        end
      end

   end

%-----------------------------------------
% Main operations loop - Fuzzy K-Means
%-----------------------------------------
while ((objective > epsilon)&(iterations < max_iter))

   for i=1:CLUST
      numerator = zeros(1,n);
      denominator = 0.0;
      for j=1:m
         numerator = numerator + membships(j,i)^2 * data(j,:);
         denominator = denominator + membships(j,i)^2;
      end
      centroids(i,:) = numerator ./ denominator;
   end

   oldships = membships;

   % Calculate the distances
   %------------------------
      for i=1:m
      for j=1:CLUST
         XsubV = data(i,:) - centroids(j,:);
         tmp = XsubV * XsubV';
         distances(i,j) =  tmp;
      end
   end

   % Calculate the  memberships
   %---------------------------
   for i=1:m
      sum_all = 0.0;
      FLAG = -1;
      for j=1:CLUST
         if( distances(i,j) ~= 0.0)
            sum_all = sum_all + 1.0/distances(i,j);
         else
            FLAG = j;
         end
      end

      for j=1:CLUST
         if(FLAG == -1)
            membships(i,j) = (1.0/distances(i,j))/sum_all;
         else
            if(j == FLAG)
```

```
                membships(i,j) = 1.0;
             else
                membships(i,j) = 0.0;
             end
          end
       end

    end


    % Calculate the objective function
    %-------------------------------
    objective = max(max(abs(oldships - membships)));
    track = [track; objective];
    iterations = iterations + 1

end

plot(track)
title('FKM - Objective function')
xlabel('Iterations')
grid



% Initialise the Fuzzy covariance matrix
%--------------------------------------
fuzz_cov =eye(n);
% Calculate the distances (exponential)
%------------------------------------
for i=1:m
   for j=1:CLUST
      P=0.0;
      for i=1:m
         P = P + membships(i,j)/m;
      end
      XsubV = data(i,:) - centroids(j,:);
      distances(i,j) = (1.0/P)*exp((XsubV * XsubV')/2);
   end
end
% Calculate the  memberships
%--------------------------
for i=1:m
   sum_all = 0.0;
   FLAG = -1;
   for j=1:CLUST
      if( distances(i,j) ~= 0.0)
         sum_all = sum_all + 1.0/distances(i,j);
      else
         FLAG = j;
      end
   end

   for j=1:CLUST
      if(FLAG == -1)
         membships(i,j) = (1.0/distances(i,j))/sum_all;
      else
         if(j == FLAG)
            membships(i,j) = 1.0;
         else
            membships(i,j) = 0.0;
         end
      end
   end
end
```

```
iterations = 0;
objective = 10;
track2=[];

%-----------------------------------------------------------
% Main operations loop - Fuzzy Maximum Likelihood Estimation
%-----------------------------------------------------------
while ((objective > epsilon)&(iterations < max_iter))

   % Calculate the  centroids
   %--------------------------
   for i=1:CLUST
      numerator = zeros(1,n);
      denominator = 0.0;
      for j=1:m
         numerator = numerator + membships(j,i)^2 * data(j,:);
         denominator = denominator + membships(j,i)^2;
      end
      centroids(i,:) = numerator ./ denominator;
   end

   % Calculate the Fuzzy covariance matrix for the centroids
   %--------------------------------------------------------
   for j=1:CLUST
      P=0.0;
      for i=1:m
         P = P + membships(i,j)/m;
      end
      denom = 0.0;
      for i=1:m
         XsubV = data(i,:) - centroids(j,:);
         fuzz_cov = fuzz_cov + membships(i,j) .* (XsubV' * XsubV);
         denom = denom + membships(i,j);
      end
      fuzz_cov = fuzz_cov ./ denom;
      sqrt_F_det = sqrt(det(fuzz_cov));
      for i=1:m
         XsubV = data(i,:)-centroids(j,:);
         distances(i,j) = (sqrt_F_det/P)*exp((XsubV * inv(fuzz_cov) * XsubV')/2);
      end
   end

   oldships = membships;

   % Calculate the  memberships
   %--------------------------
   for i=1:m
      sum_all = 0.0;
      FLAG = -1;
      for j=1:CLUST
         if( distances(i,j) ~= 0.0)
            sum_all = sum_all + 1.0/distances(i,j);
         else
            FLAG = j;
         end
      end

      for j=1:CLUST
         if(FLAG == -1)
            membships(i,j) = (1.0/distances(i,j))/sum_all;
         else
            if(j == FLAG)
               membships(i,j) = 1.0;
            else
```

```
                membships(i,j) = 0.0;
              end
            end
          end

        end

        % Calculate the objective function
        %--------------------------------
        objective = max(max(abs(oldships - membships)));
        track2 = [track2; objective];
        iterations = iterations + 1

      end

      figure
      plot(track2)
      title('FMLE - Objective function')
      xlabel('Iterations')
      grid
```

B.4 Code to Determine the Number of Hidden Nodes

```
%
% Matlab Script file - D.N.Prescott
% ================================================
% Calculation of Hidden Nodes for a Neural Net
%
% Bernie's Formula (Widrow)
%
% 10 *( [Num_ftrs +1]*Nodes + (Nodes+1)*[Num_outputs] ) < Num_data_samples
%
%  eg.
%
% 20 dimensional speech features
%  12 speakers
%  20,000 sample feature vectors
%
%   10( [20+1]*N + (N+1)*12 ) < 20,000
%
% => 33*N + 12 < 2000
%
% => N < (2000-12)/33
%
% => N < 60.24
% ==================================================

dims = 20;
num_samples = 1000;
num_classes = 12;
loops = 20;
ANNmtx = zeros(loops+1,2);
ANNmtx(1,2) =0;

for i=1:loops
   Nodes = ((num_samples/10)-num_classes)/(dims+1+num_classes);
   ANNmtx(i+1,1) = num_samples;
   ANNmtx(i+1,2) = Nodes;
   num_samples = num_samples + 1000;
end
stairs(ANNmtx(:,2))
title('Maximum number of hidden nodes for data samples, 12 classes, 20dims')
xlabel('Samples X 1000')
ylabel('Hidden nodes')
grid
```

B.5 FKM configuration file

Listed below is a typical configuration file for the UFP-ONC program. The leading numbers are not included they are for explanation only. This configuration file is for the Anderson iris data set.

```
1. % Test configuration file FKM, UFP-ONC, Neale Prescott 15OCT 93
2. iris.dat
3. %
4. iris.clust
5. iris.perfm
6. U
7. 150
8. 4
9. 2
10. 6
11. 2
12. 0.001
13. 1000
```

1. Comment line
2. Feature vector (data file) name
3. The cluster file name
4. The performance file name (NOT USED but REQUIRED FOR STARTUP)
5. The mode of operation
   - **U** for UFP-ONC, ie both FKM and FMLE
   - **F** for FKM only
6. Number of feature vectors
7. Dimension of feature vectors
8. Number of initial clusters
9. Final number of clusters
10. Fuzziness, generally use 2
11. Epsilon, the stopping criteria, must be $[0, 1]$
12. Maximum number of iterations before stopping. ie If not converged in 1000 iterations then stop.

B.6 ANN configuration file

```
datafile:   timitday02.asc
hiddenlo:   20
hiddenhi:   20
hiddenint:  20
eta:         0.30
maxerr:      0.01
maxepochs:   10000
```

Bibliography

1. Anderson, E. "The irises of the Gaspe Peninsula," *Bull. American Iris Society*, *59*:2–5 (1935).
2. Asakawa, Yoshiaki, et al. "Speech Coding using Fuzzy Vector Quantization." *International Conference on Acoustics Speech and Signal Processing*. 755–758. New York: IEEE Press, 1989.
3. Asakawa, Yoshiaki, et al. "A 2.4 KBPS Speech Coding Method Based on Fuzzy Vector Quantization." *International Conference on Acoustics Speech and Signal Processing*. 755–758. New York: IEEE Press, 1989.
4. Atal, B. S. "Automatic speaker recognition based on pitch contours," *Journal of the Acoustic Society of America*, *52*(6):1687–1697 (December 1972).
5. Bezdek, James C. "A Convergence Theorem for the Fuzzy ISODATA Clustering Algorithms," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *PAMI-2*(1):1–8 (January 1980).
6. Bezdek, James C. *Pattern Recognition with Fuzzy Objective Function Algorithms*. New York, N.Y. 10013: Plenum Press, 1981. 2nd Printing.
7. Bezdek, James C., et al. "Convergence Theorems and Fuzzy c-means : Counterexamples and Repairs," *IEEE Transactions on Systems, Man and Cybernetics*, *SMC-17*(5):873–877 (September 1987).
8. Bezdek, James C. and Sankar K. Pal, editors. *Fuzzy Models For Pattern Recognition (Selected Reprints)*, chapter 1, 1–25. New York: IEEE Press, 1992.
9. Bricker, P. D., et al. "Statistical Techniques for Talker Identification," *The Bell System Technical Journal*, *50*(4):1427–1454 (April 1971).
10. Chan, K. P. and Y. S. Cheung. "Clustering of Clusters," *Pattern Recognition*, *25*(2):211–217 (1992).
11. Colombi, Capt John. *Cepstral and Auditory Model Features for Speaker Recognition*. MS thesis, AFIT/GE/ENG/92D-11, Graduate School of Engineering, Air Force Institute of Technology (AETC), Wright-Patterson AFB OH, December 1992.
12. Cybenko, G. "Approximations by Superpositions of Sigmoidal Functions," *Mathematical Controls, Signals, and Systems* (1989).
13. Dunn, J. C. "A Fuzzy Relative to the ISODATA Process and Its Use in Detecting Compact Well-Separated Clusters," *Journal of Cybernetics*, *3*(3):32–57 (73).
14. ESPS, Washington, DC. *Entropic Signal Processing System*, 1992.
15. Fisher, R. A. "The Use of Multiple Measurements in Taxonomic Problems," *Ann Eugenics*, *7*:179–188 (1936).
16. Foil, Jerry T. and Don H. Johnson. "Text Independent Speaker Recognition," *IEEE Communications Magazine*, 22–25 (December 1983).
17. Fukunaga, Keinosuke. *Introduction to Statistical Pattern Recognition*. New York, New York: Academic Press Inc, 1972.

18. Furui, Sadaoki. "Cepstral Analysis Technique for Automatic Speaker Verification," *IEEE Transactions on Acoustics Speech and Signal Processing*, *ASSP-29*:254–272 (April 1981).

19. Gath, I. and A. B. Geva. "Unsupervised Optimal Fuzzy Clustering," *Pattern Analysis and Machine Intelligence*, *11*(7):773–781 (July 1989).

20. Gersho, Allen and Vladimir Cuperman. "Vector Quantization : A Pattern-Matching Technique for Speech Coding," *IEEE Communications Magazine*, 15–21 (December 1983).

21. Glass, Graham. *Unix for Programmers and Users A Complete Guide*. New Jersey: Prentice Hall, 1993.

22. Goldstein, Ursula G. "Speaker-Identifying features based on formant tracks," *Journal of the Acoustic Society of America*, *59*(1):176–182 (January 1976).

23. Gray, Robert M. "Vector Quantization," *IEEE ASSP Magazine*, 4–29 (April 1984).

24. Gustafson, E. E. and W. C. Kessel. "Fuzzy clustering with a fuzzy covariance matrix," *IEEE CDC*, 761–766 (1979).

25. Higgin, A. and others. *Speaker Identification and Recognition*. Final Report 88-F744200-000, ITT Aerospace/Communications Div, Nov 1991.

26. Hush, Don R. and Bill G. Horne. "Progress in Supervised Neural Networks What's New Since Lippmann?," *IEEE ASSP Magazine*, 8–39 (January 1993).

27. Keller, Capt John. *Identity Verification Through The Fusion of Face and Speaker Recognition*. MS thesis, AFIT/GE/ENG/93D-11, Graduate School of Engineering, Air Force Institute of Technology (AETC), Wright-Patterson AFB OH, December 1993.

28. Kernighan, Brian W. and Dennis M. Ritchie. *The C [ANSI C] PROGRAMMING LANGUAGE* (2nd Edition). Englewood Cliffs, New Jersey: Prentice Hall, 1989.

29. Krishnapuram, Raghu and James M. Keller. "A Possibilistic Approach to Clustering," *IEEE Transactions on Fuzzy Systems*, *1*(2):98–110 (May 1993).

30. Linde, Y., et al. "An algorithm for vector quantizer design," *IEEE Transactions on Communications*, *COM-28*:84–95 (January 1980).

31. Lippmann, Richard P. "An introduction to computing with Neural Nets," *IEEE ASSP Magazine*, 4–22 (April 1987).

32. Martin, Lt Curtis Eli. *Non-Parametric Bayes Error Estimation for UHRR RADAR Target Identification*. MS thesis, AFIT/GE/ENG/93D-26, Graduate School of Engineering, Air Force Institute of Technology (AETC), Wright-Patterson AFB OH, December 1993.

33. The MathWorks, Inc., Natick MA. *MATLAB Reference Guide*, 1992.

34. McCrae, Captain Kimberley A. *Color Image Segmentation*. MS thesis, AFIT/GE/ENG/93D-02, Graduate School of Engineering, Air Force Institute of Technology (AETC), Wright-Patterson AFB OH, December 1993.

35. Naik, Jayant M. "Speaker Verification : A Tutorial," *IEEE Communications Magazine*, 42–48 (January 1990).

36. NIST. *The DARPA TIMIT Acoustic-Phonetic Continuous Speech Corpus (TIMIT): Training and Test Data and Speech Header Software*, oct 1990.

37. Oppenheim, Alan V. and Ronald W. Schafer. *DISCRETE-TIME SIGNAL PROCESSING*. Prentice Hall, 1989.

38. O'Shaunghnessy, Douglas. "Speaker Recognition," *IEEE ASSP Magazine*, 4–17 (October 1986).

39. Pal, Sankar K. and Dwijesh Dutta Majumder. "Fuzzy Sets and Decision making Approaches in Vowel and Speaker Recognition," *IEEE Transactions on Systems, Man and Cybernetics*, 625–629 (August 1977).

40. Parsons, Thomas. *VOICE AND SPEECH PROCESSING*. New Jersey: McGraw-Hill, 1987.

41. Press, William H., et al. *Numeric Recipes in C - The Art of Scientific Computing* (2nd Edition). Cambridge UK: Press Syndicate of the University of Cambridge, 1992.

42. Rabiner, L. R. and Biing-Hwang Juang. *Fundamentals of Speech Recognition*. Englewood Cliffs, New Jersey: PTR Prentice Hall, 1993.

43. Rogers, Steven K. and Matthew Kabrisky. *An Introduction to Biological and Artificial Neural Networks*. Bellingham Washington: SPIE Press, 1991.

44. Ruspini, Enrique H. "A New Approach to Clustering," *Information Control*, *15*(1):22–32 (July 1969).

45. Spanias, Andreas S. and Frank H. Wu. "Speech Coding and Speech Recognition Technologies : A Review." *IEEE Conference on Circuits and Systems*. 572–577. New York: IEEE Press, 1991.

46. Strang, Gibert. *Linear Algebra and Its Applications* (3rd Edition). Harcourt Brace Jovanovich College Publishers, 1988.

47. Tou, J. T. and R. C. Gonzalez. *Pattern Recognition Principles*. Reading, MA: Addison-Wesley Publishing, 1974.

48. Tseng, Ho-Ping, et al. "Fuzzy Vector Quantization Applied to Hidden Markov Modeling." *International Conference on Acoustics Speech and Signal Processing*. 641–644. New York: IEEE Press, 1987.

49. Weinstein, Clifford J. "Opportunities for Advanced Speech Processing in Military Computer-Bases Systems." *Proceedings of the IEEE 79*. 1627–1639. nov 1991.

50. Woodward, J. P. and E. J. Cupples. "Selected Military Applications of Automatic Speech Recognition Technology," *IEEE Communications Magazine*, 35–41 (December 1983).

51. Zadeh, L. A. "Fuzzy Sets," *Information Control*, *8*:338–353 (1965).

Vita

Neale Prescott was born in Meningie, South Australia in 1961. He completed high school at Westminster School in Adelaide, South Australia, in 1978. Neale joined the Royal Australian Air Force in 1979 as an aircraft electrical fitter, serving with No. 9 Squadron, the Australian Contingent to the Multinational Forces and Observers in the Sinai, and No. 482 Squadron.

In 1985 he was sponsored by the RAAF to complete a Bachelor of Electronic Engineering at the Royal Melbourne Institute of Technology. On graduation in 1987, Neale received his commission as an officer in the Royal Australian Air Force.

In 1988, he was posted to No.3 Aircraft Depot as OIC Electrical Workshop and Base Calibration Centre.

From 1989 until coming to AFIT in 1992, Neale managed F111 maintenance for the RAAF Strike Reconnaissance Group and Iroquois helicopter maintenance for the Army Aviation Regiment at No.501 Wing, Queensland, Australia.

Permanent address: 14 Illawong Way
Karana Downs, Queensland
AUSTRALIA 4306